Jan Loman

Generaattori energia-alan tasepoikkeama-raporttien tuottamiseksi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

2.5.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Jan Loman Generaattori energia-alan tasepoikkeamaraporttien tuottamiseksi 51 sivua + 3 liitettä 2.5.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	lehtori Vesa Ollikainen lehtori Jussi Alhorinne järjestelmäpäällikkö Markku Kuurne

Tässä insinöörityössä tehtiin raporttigeneraattori, joka tuottaa tasepoikkeamaraportteja Rejlers Oy:n tuotantokäyttöön, sekä pohdittiin Riak-tietokannan soveltuvuutta Rejlers Oy:n ja raporttigeneraattorin tietolähteeksi. Tarve generaattorille syntyi, kun Energiateollisuus ry:n asettamat uudet menettelyohjeet vaativat sen alaisuuteen kuuluvilta toimijoilta tasepoikkeamaraporttien tuottamista. Työn tarkoitus oli helpottaa ja nopeuttaa tasepoikkeamaraporttien tuottamista sekä tuottaa tietoa Riakin soveltuvuudesta raporttigeneraattorin ja Rejlers Oy:n tietolähteeksi.

Työssä lähdetään liikkeelle esittelemällä työn tavoitteet sekä käytetyt teknologiat ja menetelmät, jotta lukijan olisi helpompi ymmärtää ohjelman määrittelyä ja toteutusta. Tämän jälkeen kuvataan työn toiminnallinen ja tekninen määrittely, jotka avaavat lukijalle ohjelman rakennetta. Määrittelyjen jälkeen esitellään raporttigeneraattorin toteutus sekä pohditaan Riakin soveltuvuutta Rejlers Oy:n ja raporttigeneraattorin tietolähteeksi. Lopuksi tehdään yhteenveto työstä ja työn tuloksista.

Työn tuloksena syntyi lähes tuotantokäyttöön soveltuva Energiateollisuus ry:n formaattien mukaisia tasepoikkeamaraportteja tuottava raporttigeneraattori. Raporttigeneraattoriin ei aikataulusyistä saatu toteutettua tasevirhetuntitiedotraportin generointia, joten raporttigeneraattori ei sovellu vielä täysin tuotantokäyttöön.

Työn tuloksena saatiin myös tietoa siitä, että Riak soveltuu hyvin Rejlers Oy:n ja raporttigeneraattorin tietolähteeksi, vaikka Riak aiheuttaa raporttigeneraattorin toteutukseen vähän lisää töitä.

Avainsanat	PostgreSQL, Raporttigeneraattori, Riak, Tasevirhe.

Author(s) Title Number of Pages Date	Jan Loman Generator that produces energy balance deviation reports 51 pages + 3 appendices 2 May 2013
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software engineering
Instructor(s)	Vesa Ollikainen, Senior Lecturer Jussi Alhorinne, Senior Lecturer Markku Kuurne, System manager

In this thesis were done a report generator that produces balance error reports for Rejlers Ltd.´s production use and considered suitability of the Riak database as a data source of Rejlers Ltd. and report generator. The need for generator was born when new guidelines for handling balance error reports were set by Finnish Energy Industries. The objective of this thesis were to make producing of balance error reports faster and easier and produce information about suitability of Riak database as a data source of Rejlers Ltd and report generator.

This thesis is started by describing the objectives and used technologies and methods, so that would be easier for a reader to understand the definition and the implementation of the report generator program. After that, the functional and the technical specification is described which opens up the program's structure to the reader. After the specifications the implementation is presented and the suitability of the Riak was considered. At the end summary about the thesis and thesis' results are made.

The result of this thesis was the almost suitable balance error report generator for the production use which follows the formats made by Finnish Energy Industries. Also as a result of this thesis the information about the suitability of the Riak was got.

Keywords	PostgreSQL, Report generator, Riak, Balance error

Sisällys

Lyhenteet

1 Johdanto 1

2 Tavoitteet 2

3 Teknologiat ja menetelmät 3

3.1 Riak 3
3.1.1 Rakenne 3
3.1.2 Kyselyt 7
3.1.3 Linkit 10

3.2 PostgreSQL 13

3.3 Suunnittelumallit 13
3.3.1 MVC-malli (Model-View-Controller pattern) 14
3.3.2 Luontimallit 15
3.3.3 Tehdasfunktio (Factory Method) 15

4 Toiminnallinen määrittely 17

4.1 Vaatimukset 17

4.2 Toiminnot 18

5 Tekninen määrittely 26

5.1 Smac API -rajapinta 26

5.2 Ohjelmointikieli 27

5.3 Ohjelmointiympäristö 27

5.4 Arkkitehtuuri 28
5.4.1 Ulkoiset komponentit 28
5.4.2 Luokkakaavio 29

5.5 Tärkeimmät kirjastot 31
5.5.1 Apache POI 31
5.5.2 Apache http Client 32

5.5.3 PostgreSQL JDBC Driver 32
5.5.4 JavaMail API 32

6 Raporttigeneraattorin toteutus 32

6.1 Raporttigeneraattorin hallinnan toteutus 32
6.1.1 Help-valikko 33
6.1.2 Asetukset-valikko 35
6.1.3 File-valikko 35

6.2 Vertailuraportin toteutus 35

6.3 Tasevirheiden tuntitiedot -raportin toteutus 40

7 Testaus 42

8 Riak-tietokannan soveltuvuus Rejlers Oy:n ja raporttigeneraattorin tietolähteeksi 44

9 Yhteenveto 48

Lähteet 51

Liitteet

Liite 1. Tasevirhettä koskevien raporttien formaattien määrittely dokumentti

Liite 2. Tasevertailu-raportin malliesimerkki

Liite 3. Tasevirhetuntitiedot-raportin malliesimerkki

Lyhenteet

CSV	Comma-separated values on tiedostomuoto, jolla tallennetaan pilkuilla eroteltua taulukkomuotoista tietoa tekstitiedostoon.
GNU GPL	Gnu General Public License on lisenssi, jota käytetään vapaiden ohjelmistojen julkaisemiseen.
javac	Java-ohjelmointikielen kääntäjä. Se on tietokoneohjelma, joka luo ohjelman lähdekoodin perusteella konekielisen ajettavan binääritiedoston.
JDK	Java Development Kit on Javan ohjelmistokehityspaketti.
JRE	Java Runtime Environment on Javan ajoympäristö, jota tarvitaan käännettyjen ohjelmien ajamiseen.
JSON	JavaScript Object Notation on tiedonsiirtomuoto.
JVM	Java Virtual Machine sisältää ohjelman ajonaikaisen käännöksen konekielelle.
NoSQL	Tietokanta, joka ei käytä SQL-kieltä tietokannan hallintaan.
vnode	Virtuaalisolmu eli prosessi, joka hallinnoi yhtä bucketin kokonaislukuavaruuden osaa.

1 Johdanto

Työ- ja elinkeinoministeriön sähkökauppojen selvittämiseen liittyvästä tiedonvaihdosta antaman asetuksen seurauksena taseikkuna eli aikaväli, jolloin sähkönmittauksen mittaustulos voi muuttua, lyheni kuukaudesta 14 vuorokauteen vuoden 2011 alusta alkaen. Tämä tarkoittaa huomattavasti tiukempaa aikataulua taseisiin jääneiden virheiden selvittämiseen.

Samaan aikaan sähkönmittauksessa on ollut käynnissä suuri muutos. Sähkönmittauksessa ollaan siirtymässä kuukausittaisesta mittaamisesta tuntimittaukseen, joka tarkoittaa noin kolmen miljoonan käyttöpaikan siirtymistä tuntimittaukseen viimeistään 2014 mennessä ja tiedon räjähdysmäistä kasvua. Aikaisemmin käyttöpaikalta on kerätty tietoa kerran kuukaudessa, kun taas tuntimittauksessa käyttöpaikalta kerätään tietoja kerran tunnissa. Tietoa tulee noin 700 kertaa enemmän kuin aikaisemmin.

Nämä kaksi muutosta, kasvava käsiteltävän tiedon määrä sekä tiukempi aikataulu, ovat aiheuttaneet vaaran, että jakeluverkon taseisiin saattaa jäädä suurempi määrä virheitä kuin aikaisemmin. Tästä syntyneen tarpeen vuoksi Energiateollisuus ry on laatinut selkeämmät ja ajantasaiset menettelyohjeet taseisiin jääneiden virheiden korjaukselle toimijoiden välillä taseiden sulkeutumisen jälkeen.

Tässä opinnäytetyössä esitellään automatisoitu ratkaisu eli raporttigeneraattori, jolla tuotetaan Energiateollisuus ry:n laatimia formaatteja noudattavia tasevirhettä koskevia raportteja Rejlers Oy:lle sekä pohditaan, onko tietokantojen markkinoilla suuressa kasvussa oleviin NoSQL-tietokantoihin lukeutuva Riak-tietokanta hyvä ratkaisu raporttigeneraattorin ja Rejlers Oy:n tietolähteeksi. Rejlers Oy on asiantuntijaorganisaatio, joka tarjoaa suunnittelu- ja konsultointipalveluita sekä projektitoimituksia teollisuuden, energian, rakentamisen, kiinteistöjen sekä infran asiakkaille [1].

Ratkaisussa tuotettavia tasevirhettä koskevia raportteja on kahdenlaisia. Toisessa raportissa vertaillaan myyjän ja verkonhaltijan tietoja tasevirheiden löytämiseksi, kun taas toisen raportin avulla verkkoyhtiöt ilmoittavat myyjälle taseiden sulkeutumisen jälkeen korjatut käyttöpaikkakohtaiset tuntiaikasarjat sekä näiden aiheuttamat euromääräiset muutokset tunneittain ajanjaksolta, jota raportti käsittelee. Aiemmin Rejlers Oy on tuottanut vastaavanlaisia tasevirhettä koskevia raportteja manuaalisesti. Tämä prosessi on

ollut paljon aikaa vievä ja työläs. Siksi raporttien tuottamisen automatisointi on tarpeellinen ja tervetullut uudistus.

Työssä lähdetään liikkeelle kuvaamalla työn tavoitteet, jottei jäisi epäselväksi, mihin työssä pyritään ja mitä on tarkoitus saada aikaiseksi. Tavoitteiden jälkeen luvussa 2 esitellään teknologiat ja menetelmät, joilla tämä työ on toteutettu. Luvuissa 4, 5 ja 6 kuvataan raporttigeneraattorin suunnittelu, toteutus ja testaus. Lopuksi arvioidaan Riak-tiedonhallintajärjestelmän soveltuvuutta raporttigeneraattorin ja Rejlers Oy:n tietolähteeksi.

2 Tavoitteet

Tämä työ jakautuu kahteen eri tavoitteeseen. Ensimmäinen ja tärkeämpi tavoite on saada itse raporttigeneraattori valmiiksi tuotantokäyttöön. Jotta raporttigeneraattori täyttäisi tämän tavoitteen, sen tulisi tuottaa annettujen formaattien mukaisia raportteja. Tavoite on tärkeä sen takia, että raporttigeneraattorin tuottamia raportteja tarvitaan kevään 2013 aikana. Ilman raporttigeneraattoria raportit täytyy tehdä käsin, mikä vie erittäin paljon aikaa ja resursseja.

Toinen tavoitteista on Riak-tiedonhallintajärjestelmän soveltuvuuden arviointi raporttigeneraattorin ja Rejlers Oy:n tietolähteeksi. Tässä tavoitteessa pyritään vastaamaan muun muassa kysymyksiin, onko Riakin käyttö kannattavaa sovelluksen rakenteen näkökulmasta, olisiko parempi ratkaisu käyttää vain olemassa olevaa PostgreSQL-tietokantaa sekä nopeuttaako Riak sovelluksen käyttöä. Tämän tavoitteen täyttämiseksi tarvitaan erilaisia vertailuja, kuten kustannus- ja suorituskykyvertailuja, Riakin ja PostgreSQL:n välillä ja käytännön kokemuksia molempien tietokantojen käytöstä. Tavoitteen täyttyessä Rejlers Oy saa paljon arvokasta tietoa Riak-tietokannan soveltuvuudesta, mikä voi vaikuttaa yrityksen tekemiin tietokantaratkaisuihin.

3 Teknologiat ja menetelmät

3.1 Riak

Riak on avoimen lähdekoodin NoSQL-tietokanta, joka on kirjoitettu Erlang-ohjelmointikielellä. Riakin kotisivujen [2] mukaan Riak eroaa perinteisistä relaatiotietokannoista ja muista NoSQL eli ei-relaatiotietokannoista. Sivustolla Riakin vahvuuksiksi kuvataan sen ennakoitavuutta, saatavuutta, skaalautuvuutta ja vikasietoisuutta sekä sen kerrotaan olevan suunniteltu toimimaan hajautetuissa ympäristöissä, kuten hyvin suuressa kasvussa olevissa pilvipalveluissa. Sivustolla sanotaankin, että suunnittelunsa ansiosta Riak on hyvin vikasietoinen.

Riak on saanut alkunsa, kun Basho Technologies kehitti Riak-tietokantaa osaksi sovellusta. Basho Technologies kiinnostui kuitenkin enemmän tietokantateknologiasta ja päätti keskittää toiminnan Riakin ympärille eikä itse sovelluksen, johon sen oli alun perin pitänyt tulla vain osaksi.

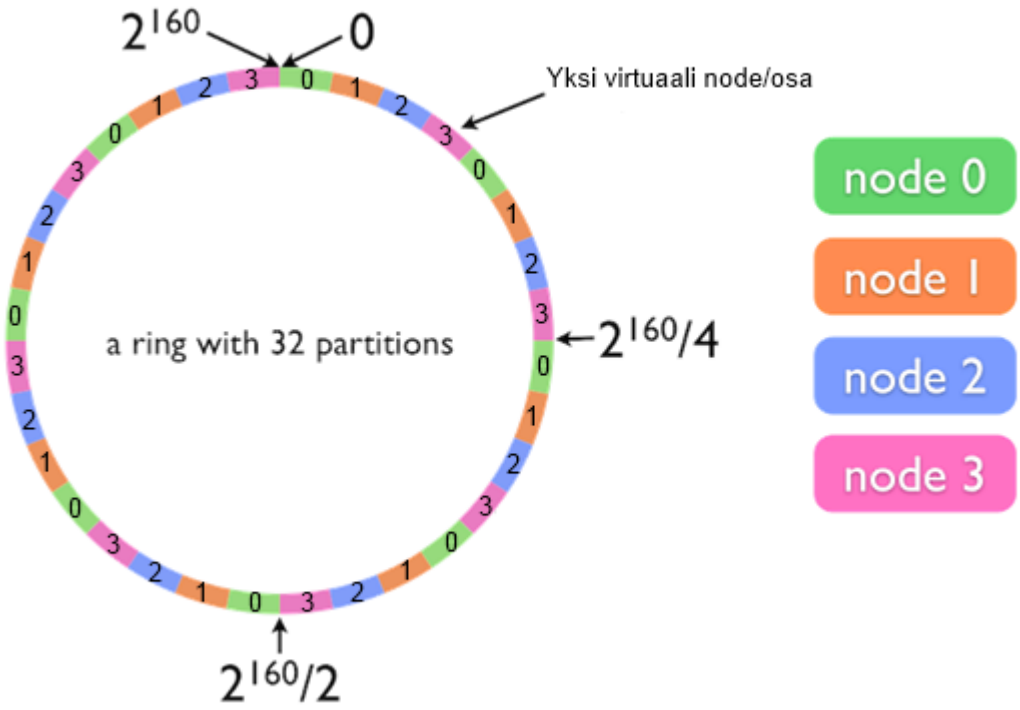
Riakin kotisivuilla [2] kerrotaan, että Riak on saanut vaikutteita tohtori Eric Brewerin CAP-teoreemasta, jossa todetaan, että jaetun tietokonejärjestelmän on mahdotonta tarjota samanaikaisesti johdonmukaisuutta, saatavuutta ja osiosuvaitsevaisuutta. Saatavuudella tarkoitetaan sitä, että tieto on saatavilla, vaikka osa tietokannan nodeista eli solmuista on epäkunnossa. Osiosuvaitsevaisuudella tarkoitetaan sitä, että solmut voivat olla fyysisesti erotettu toisistaan ja silti niille pitää pystyä suorittamaan luku- ja kirjoituspyyntöjä. Johdonmukaisessa tietokannassa yhteen solmuun päivitetty tieto on välittömästi saatavilla muissa solmuissa. Sivustolla mainitaan myös, että Riak on saanut paljon vaikutteita Amazon's Dynamo paperista, joka on Amazon.com:in ydinpalveluissa käytetty avain-arvovarastojärjestelmä.

3.1.1 Rakenne

Riak rakentuu nimiavaruuksista eli bucketeista, joihin avain-arvo-parit tallennetaan. Yhteen buckettiin tallennetaan usein yhteen asiaan liittyvät avain-arvo-parit, kuten esimerkiksi tietystä paikasta otetut kuvat tallennetaan samaan buckettiin. Jokainen buckettiin tallennettu avain-arvo-pari hajautetaan ja hajauttamisesta syntynyt arvo sijoitetaan 160-bittiseen kokonaislukuavaruuteen eli klusteriin. Klusteri koostuu fyysisistä

koneista ja sen visualisoimiseen käytetään usein rengasta (kuva 1), jotta saadaan havainnollistettua, mikä tieto talletetaan mihinkin fyysiseen koneeseen eli solmuun. Riakin kaikki solmut ovat samanarvoisia. Yksikään solmu ei ole isäntä tai orja kuten useimmissa relaatiotietokannoissa, joissa isäntäsolmu ohjaa kirjoitusoperaatioita. Relaatiotietokannassa isäntäsolmu-virhetilanteissa kirjoitukset epäonnistuvat, kunnes virhe on ratkaistu, kun taas Riakissa kirjoitusoperaatiot onnistuvat solmun virhetilanteessa, koska kaikki solmut voivat suorittaa kirjoitusoperaatioita.

Riak jakaa 160-bittisen kokonaislukuavaruuden eli klusterin yhtä suuriksi osiksi. Oletuksena avaruus jaetaan 64 osaan. Jokainen osa vastaa tietystä renkaan arvoalueesta. Jokaista avaruuden osaa hallinnoi prosessi, jota kutsutaan vnodeksi eli virtuaalisolmuksi. Fyysiset koneet eli solmut, joita voi lisätä ja poistaa dynaamisesti, jakavat vastuunsa tasaisesti virtuaalisolmuille. Jokainen fyysinen kone vastaa sen virtuaalisolmujen edustamista avain-arvo-pareista. Seuraavassa kuvassa (kuva 1) on esitelty tilanne, jossa on 4 solmua käsittävä klusteri, jonka kokonaislukuavaruus on jaettu 32:een yhtä suureen osaan.



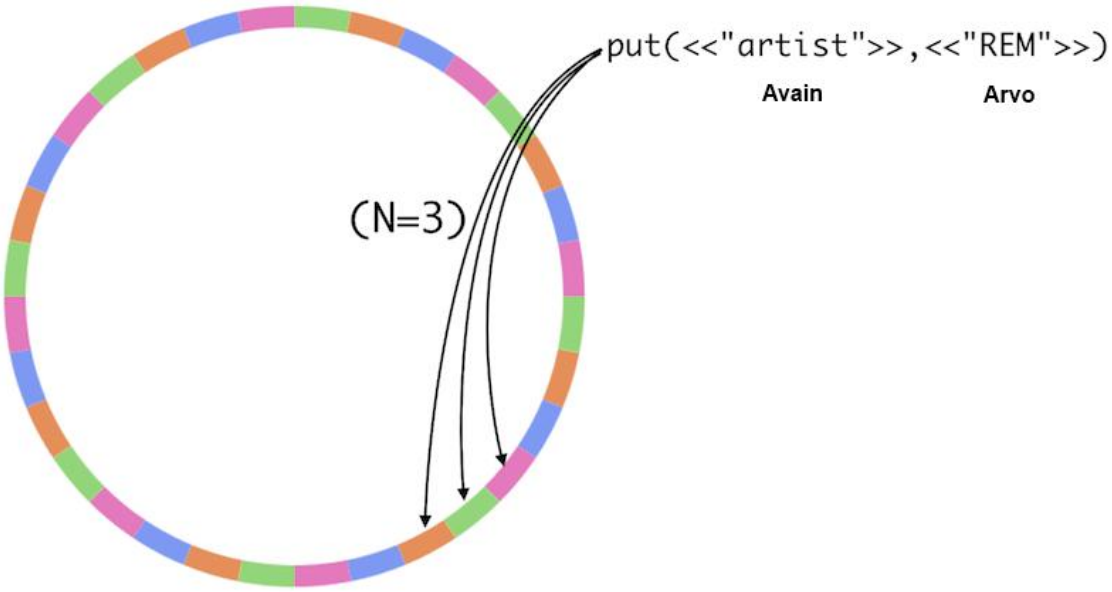
Kuva 1. Klusterin kokonaislukuavaruus jaettuna 32 yhtä suureen osaan. [3]

Kuten kuvasta 1 näkee, niin kokonaislukuavaruuden 32 osaa on jaettu tasaisesti kaikille solmuille, jolloin jokaisella solmulla on kahdeksan osaa ja samalla myös kahdeksan virtuaalisolmua. Jokainen solmu vastaa sen virtuaalisolmujen kattamista arvoalueiden avain-arvo-pareista. Esimerkiksi kuvan solmu node 0 vastaa kaikista arvoalueiden avain-arvo-pareista, jotka sijoittuvat renkaan sille varatuille (vihreille) alueille.

Tässä vaiheessa on puhuttu vain Riakiin tallennettavista avain-arvo-pareista. Tämä ei kerro paljoa siitä, mitä Riakiin voi tallentaa. Riakin kotisivujen [2] mukaan Riakiin voi tallentaa mitä tahansa. Kuitenkin sivustolla mainitaan vain tekstin, kuvien, JSON/XML/HTML-dokumenttien, käyttäjä-/tapahtumatietojen, varmuuskopioiden ja lokitiedostojen tallentaminen.

Se mitä Riakiin tallennetaan, ei välttämättä anna koko kuvaa Riakin objektien rakenteesta eli siitä, miten objektit ovat yhteydessä toisiinsa. Objektia tallennettaessa objektille määritellään tyyppi, arvo, bucket sekä avain. Esimerkiksi tallennettaessa vaikka Testi-yhtyeen kappale ”Tämä on testi” musiikki-kirjastoon tämä tapahtuu määrittelemällä tallennettavan objektin tyypiksi mp3-tiedosto ja arvoksi itse musiikkitiedosto. Bucketiksi valitaan musiikkikirjasto ja avaimeksi yhtyeen nimi Testi. Tallennus tapahtuisi komennolla PUT /riak/musiikkikirjasto/Testi.

Seuraavaksi siirrytään aiheeseen nimeltä saatavuus, joka on yksi Riakin vahvuuksista. Saatavuus tarkoittaa sitä, että tieto on saatavilla, vaikka jokin tai jotkin tietokannan solmuista olisivat epäkunnossa. Saatavuuden hahmottamista helpottaa alla oleva kuva (kuva 2).



Kuva 2. Talletettavan objektin kopioituminen kokonaislukuavaruuden osille. [3]

Riakin saatavuus perustuu siihen, että tallennettu objekti kirjoitetaan usealle kokonaislukuavaruuden osalle (kuva 2). Jolloin solmun tuhoutuessa toiset solmut pystyvät esittämään tiedot samalla tavalla kuin solmua ei olisi tuhoutunut ollenkaan. Sitä kuinka monelle osalle objekti kirjoitetaan eli kuinka monta kopiota tehdään, merkataan kirjaimella N, jota on mahdollista muuttaa bucket-kohtaisesti. Talletetusta objektista tehtyjen kopioiden määrä vaikuttaa tietokannan vikasietoisuuteen.

Riakin vikasietoisuus voidaan määrittää operaatiokohtaisesti. Operaatiolle voidaan määrittää kopioiden määrä, joka tarvitaan onnistuneeseen kirjoitus- (W) tai lukuoperaatioon (R). Lisäksi vikasietoisuuden määrittämiseen tarvitaan bucket-kohtainen kopioiden määrä (N). Seuraavaksi on esimerkki operaatiosta, jonka kirjoitus- ja lukuoperaation vikasietoisuus on määritelty.

N = kopioiden määrä

W = vastaa kopioiden määrää, joka tarvitaan onnistuneeseen kirjoitusoperaatioon.

R = vastaa kopioiden määrää, joka tarvitaan onnistuneeseen lukuoperaatioon.

N – R = lukuoperaation virheensietokyky

N – W = kirjoitusoperaation virheensietokyky

Esimerkkitilanne operaatiosta:

N = 4, W = 3, R = 1

4 – 3 = 1 → 1 solmu voi olla epäkunnossa ja Riak voi silti suorittaa kirjoitusoperaatioita.

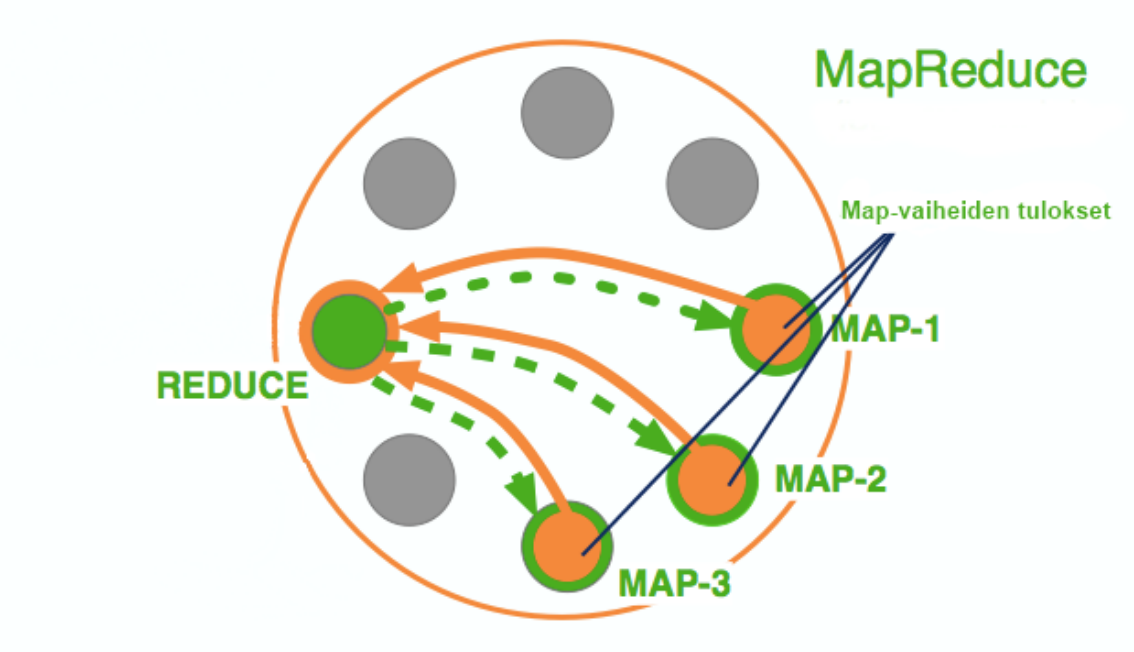
4 – 1 = 3 → 3 solmua voi olla epäkunnossa ja Riak voi silti suorittaa lukuoperaatioita.

3.1.2 Kyselyt

MapReduce on Riakin keskeinen kyselytekniikka, jossa kysely jaetaan ympäri hajautettua järjestelmää, jolloin kysely pystyy hyödyntämään hajautetun järjestelmän rinnakkaiskäsittelyn tehokkuutta. Tätä tekniikkaa käytetään useimmiten dokumenttien suodattamiseen, sanojen laskemiseen ja tutkittavaan aineistoon liittyvien linkkien poimimiseen.

MapReduce-kysely koostuu bucket-avainparilistasta sekä vaihelistasta, jonka vaiheita voivat olla map, reduce ja link. Map-vaiheessa solmusta, johon käyttäjä otti yhteyden kyselyä tehdessä, tulee koordinaattorisolmu. Se ohjaa bucket-avainparit ja kyselyn toiminnon virtuaalisolmuille, joissa kyselyn toiminto suoritetaan. Kun toiminto on suoritettu, niin virtuaalisolmu lähettää tuloksen takaisin koordinaattorisolmulle, joka ketjuttaa tulokset. Tämän jälkeen siirrytään vaihelistan seuraavaan vaiheeseen.

Jos eri vaiheessa kerätyt tulokset halutaan kasata yhdeksi tulokseksi, niin se voidaan tehdä Reduce-vaiheen avulla, joka yhdistää vaiheiden tulokset yhdeksi tulokseksi. Tämä saatu tulos voidaan muotoilla esimerkiksi sisältämään map-vaiheiden tulokset, jotka sisältävät tietyn sanan. Reduce-vaiheen toimintaa havainnollistaa kuva 3, jossa on esimerkki Reduce-vaiheesta, joka kerää kolmen Map-vaiheen tulokset yhdeksi tulokseksi.



Kuva 3. MapReduce-kyselyn Map-vaiheiden tuloksien kokoaminen Reduce-vaiheen avulla. [3]

Kuvasta 3 nähdään, kuinka eri Map-vaiheiden tulokset siirtyvät Reduce-vaiheelle, joka kokoaa keräämänsä vaiheiden tulokset yhdeksi tulokseksi. Jotta MapReducen käyttö ei jäisi vain teorian varaan, niin seuraavaksi käydään läpi esimerkki MapReducen käytöstä. Esimerkissä on käytetty pohjana Riakin dokumenttisivustolta [3] löytyvää esimerkkiä.

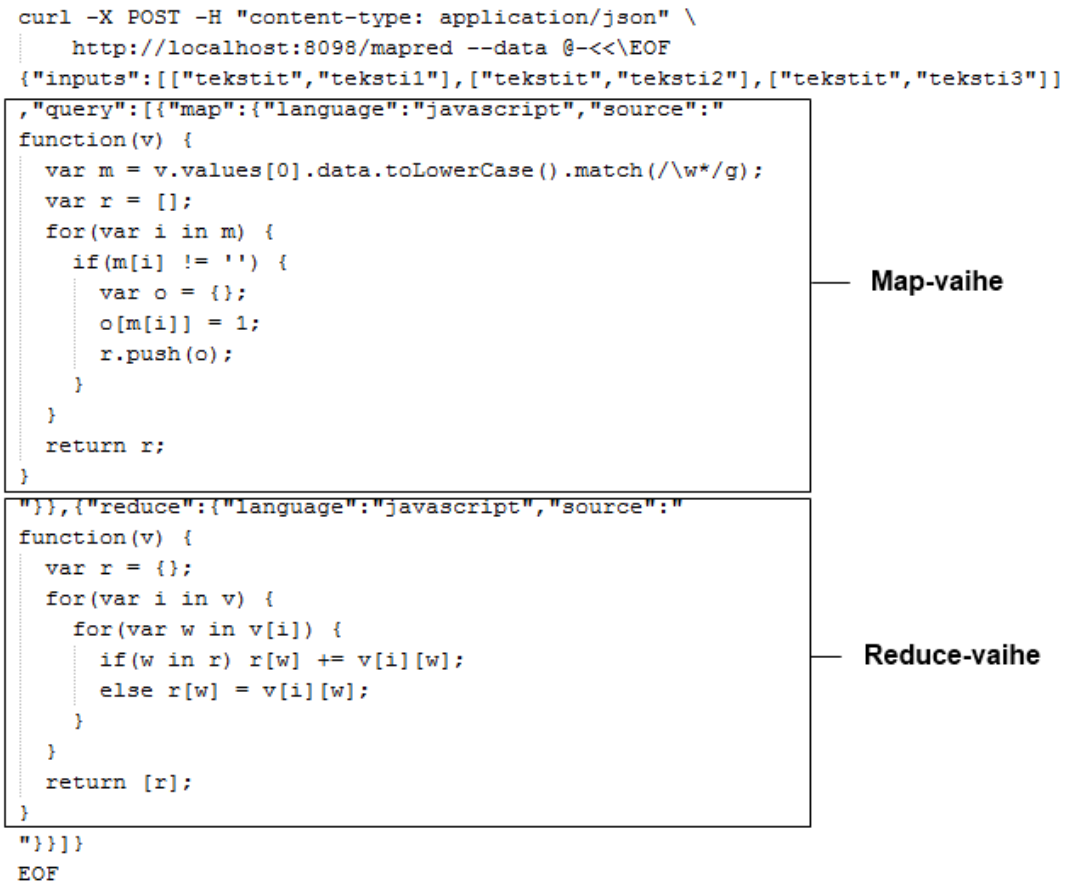
```
curl -XPUT -H "content-type: text/plain" \
    http://localhost:8098/riak/tekstit/teksti1 --data-binary @-<<\EOF
Tama on esimerkin ensimmainen teksti, josta lasketaan sanojen esiintymiskertoja.
Tekstilla ei ole merkitysta. Tama on kirjoitettu vain, jotta esimerkissa olisi jotain sisaltoa.
EOF

curl -XPUT -H "content-type: text/plain" \
    http://localhost:8098/riak/tekstit/teksti2 --data-binary @-<<\EOF
Esimerkissa kaikki tekstit ovat talletettu text/plain-muodossa, kuten tamakin teksti.
Kaikki esimerkin tekstit sijaitsevat tekstit bucketissa, josta ne kaydaan lapi mapreducen avulla.
EOF

curl -XPUT -H "content-type: text/plain" \
    http://localhost:8098/riak/tekstit/teksti3 --data-binary @-<<\EOF
Tassa on kolmannen tekstin sisalto ja se ei ole kovinkaan pitka, mutta ei se haittaa.
EOF
```

Kuva 4. MapReduce-työn aineisto

Yllä olevassa kuvassa 4 on MapReduce-työn aineisto. Aineisto koostuu kolmesta tekstistä, jotka on sijoitettu buckettiin nimeltä tekstit ja joiden avaimet ovat teksti1, teksti2 ja teksti3. Seuraavassa kuvassa 5 on MapReduce-työn Map- ja Reduce-vaiheet.



Kuva 5. MapReduce-työn Map- ja Reduce-vaiheet

Kuvan 5 ylempi nelikulmiolla erotettu alue on työn Map-vaihe. Siinä luodaan lista JSON-objekteja, joita on yksi jokaista aineistosta löytyvää sanaa kohden. Jokaisen objektin avain on sen edustama aineiston sana ja sen arvona on kokonaisluku yksi. Alempi nelikulmiolla erotettu alue on taas työn Reduce-vaihe. Siinä käydään läpi Map-vaiheen jokainen JSON-objekti sekä luodaan uusi objekti. Tämä uusi objekti on lista ja se luodaan käymällä läpi kaikki JSON-objektit ja lisäämällä listaan JSON-objektin avain ja arvo, mikäli avainta ei vielä ole listassa, niin avain lisätään listaan ja sen arvoksi tulee JSON-objektin arvo. Jos avain löytyy jo listasta niin, sen arvoa kasvatetaan JSON-objektin arvolla. Lopulta saadaan seuraavassa kuvassa 6 esitelty MapReduce-työn tuottama tulos.

[{"tama":2,"esimerkissa":2,"kaikki":2,"on":3,"tekstit":3,"esimerkin":2,"ovat":1,"ensimmai
nen":1,"talletettu":1,"teksti":2,"tassa":1,"text":1,"josta":2,"plain":1,"lasketaan":1,"ko
lmannen":1,"muodossa":1,"sanojen":1,"tekstin":1,"kuten":1,"esiintymiskertoja":1,"sisalto"
:1,"tamakin":1,"tekstilla":1,"ja":1,"ei":3,"se":2,"ole":2,"merkitysta":1,"kovinkaan":1,"s
ijaitsevat":1,"pitka":1,"kirjoitettu":1,"mutta":1,"bucketissa":1,"vain":1,"jotta":1,"ne":
1,"haittaa":1,"kaydaan":1,"olisi":1,"lapi":1,"jotain":1,"mapreducen":1,"sisaltoa":1,"avul
la":1}][root@riaktesti riak]#

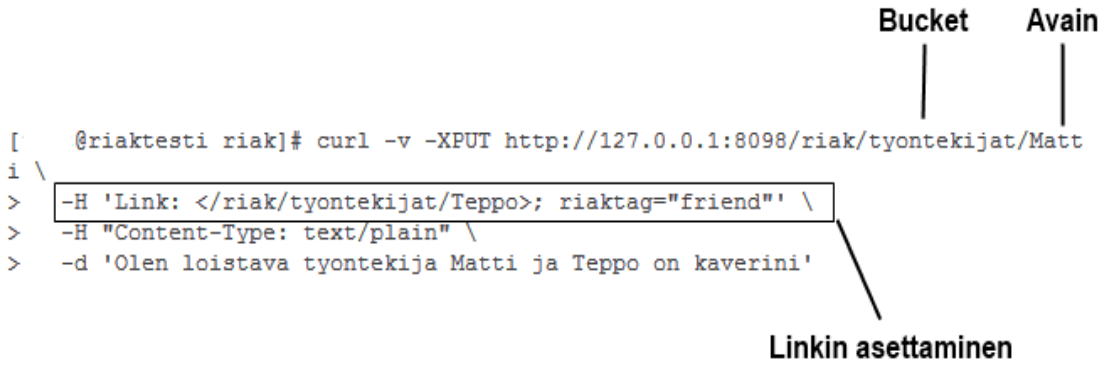
Kuva 6. MapReduce-työn tulos

Tuloksesta nähdään, kuinka monta kertaa aineiston mikäkin sana esiintyy aineistossa. Kuten kuvasta voi nähdä, niin esimerkiksi sana text esiintyy aineistossa kerran ja sana teksti kaksi kertaa.

MapReduce ei kuitenkaan ole Riakin ainoa kyselytekniikka. Riakissa onnistuvat aivan tavalliset kyselyt kuten tallennus, poisto ja haku. Nämä tavalliset kyselyt ovat muotoa haluttu toiminto /riak/bucket/avain. Toimintoja ovat PUT (tallennus), GET (haku) ja DELETE (poisto). Muita Riakin kyselytapoja ovat muun muassa Link Walking, jolla saadaan kyseltyä myös avaimeen liitetyt linkit sekä Secondary Indexes, jolla voi liittää lisätietoja avaimelle.

3.1.3 Linkit

Riakissa tiedot eivät ole samalla tavalla yhteydessä toisiinsa kuin relaatiotietokannoissa. Riakissa käytetään relaatioiden sijasta linkkejä, kun objektiin halutaan liittää siihen kuuluvaa tietoa. Linkit ovat yksisuuntaisia suhteita objektien välillä. Kun objektille on asetettu linkki toiseen objektiin, niin kyselyä suoritettaessa voidaan käyttää toimintoa, jossa kävellään eli siirrytään objektin linkkiä pitkin linkitettyyn objektiin. Tästä linkkiä pitkin kävelemisestä toiminto saa myös nimensä ”Link Walking”. Seuraavaksi käydään läpi vaihe vaiheelta esimerkki linkin lisäämisestä objektien välille. Esimerkki pohjautuu Riakin dokumenttisivuston [3] esimerkkiin.



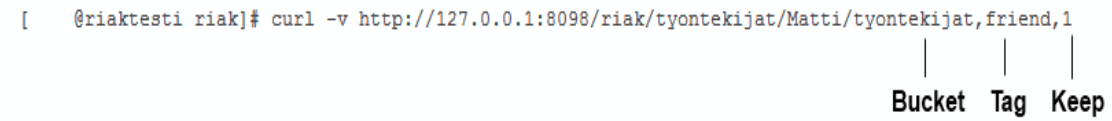
Kuva 7. Linkin asettaminen

Yllä olevassa kuvassa 7 tyontekijat-bucketissa oleva Teppo-avain liitetään ”friend”-linkillä tyontekijat-bucketin Matti-avaimeen. Linkin asettaminen toiseen objektiin ei kuitenkaan tarkoita sitä, että tämä toinen objekti olisi jo talletettuna tietokantaan. Jotta linkistä olisi jotain hyötyä, niin seuraavaksi tallennetaan kantaan objekti Teppo, johon yllä lisätty linkki osoittaa (kuva 8).

```
[ @riaktesti riak]# curl -v -XPUT http://127.0.0.1:8098/riak/tyontekijat/Teppo \
> -H "Content-Type: text/plain" \
> -d 'Minä olen tyontekijä Teppo'
```

Kuva 8. Talletetaan Teppo-objekti

Nyt ollaan tilanteessa, jossa Matti-objekti on tallennettu linkillä, joka osoittaa samassa bucketissa olevaan Teppo-objektiin. Linkki on näin valmis käytettäväksi kyselyissä. Tapa, jolla linkkiä käytetään kyselyissä, on aiemmin mainittu Link Walking, jossa objetista siirrytään toiseen linkkiä pitkin. Link Walking -kyselyn rakenne koostuu kolmesta osasta, jotka ovat bucket, tag ja keep. Rakenteen bucket-osa rajaa kyselyn koskemaan tiettyä buckettia, tag-osa määrittää, mitä linkkityyppiä haetaan ja keep-osa määrittää, ollaanko vaiheen tuloksesta kiinnostuneita eli liitetäänkö vaiheen tulos kyselyn tulokseen. Kuvassa 9 on Link Walking -kysely, jossa käytetään edellisen esimerkin Matti-objektia.



Kuva 9. Link Walking -kysely

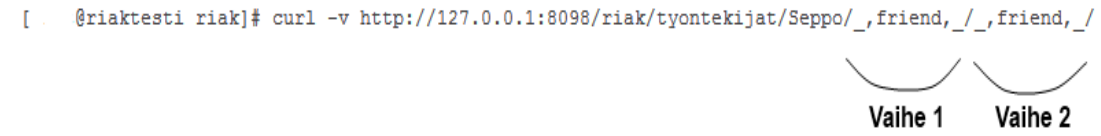
Kyselyssä haetaan tyontekijat-bucketissa sijaitsevia objekteja, jotka on liitetty friend-linkillä Matti-objektiin. Kyselyn keep-osassa on valittu numero 1, joka tarkoittaa sitä, että vaiheen tuloksesta ollaan kiinnostuneita. Jos numero yhden tilalla olisi numero nolla, niin silloin tämä kysely olisi turha, koska se ei palauttaisi yhtään tulosta.

Yhdessä Link Walking -kyselyssä voidaan kävellä useita linkkejä pitkin, jolloin kyselyn rakenteen keep-osalle tulee käyttöä. Lisäämällä edellisten esimerkkien objekteihin kolmas objekti (kuva 10), saadaan esiteltyä keep-osan toiminta.

```
[   @riaktesti riak]# curl -v -XPUT http://127.0.0.1:8098/riak/tyontkeijat/Seppo \
>  -H 'Link: </riak/tyontekijat/Matti>; riaktag="friend"' \
>  -H "Content-Type: text/plain" \
>  -d 'Minä olen työntekijä Seppo ja Matti on kaverini'
```

Kuva 10. Talletetaan Seppo-objekti, jolla on friend-linkki osoitettuna Matti-objektiin.

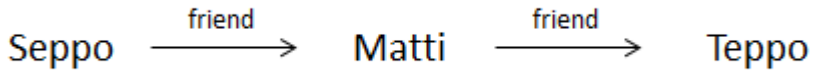
Jos halutaan tehdä kysely, jossa halutaan päästä uudesta Seppo-objektista Teppo-objektiin, johon Seppo-objektilla ei ole linkkiä, niin se onnistuu kuvassa 11 esitetyllä tavalla.



Kuva 11. Link Walking -kysely, jossa kävellään kahta linkkiä pitkin.

Kuvan kyselyn ensimmäisessä vaiheessa haetaan kaikista bucketeista objekteja, joilla on friend-linkki liitettynä Seppo-objektiin. Toisessa vaiheessa haetaan kaikista bucke-

teista objekteja, jotka on liitetty friend-linkillä ensimmäisen vaiheen tuloksena saatuihin objekteihin. Kyselyn tuloksena saadaan Teppo-objekti, johon Seppo-objektin friend-linkin osoittaman Matti-objektin friend-linkki osoittaa. Jotta tietojen linkittyminen ei jäisi epäselväksi, niin seuraavassa kuvassa 12 on esitetty, miten objektit ovat linkitettyinä toisiinsa. Kuvan nuolet edustavat linkkejä ja nuolien suunnat kuvaavat, mistä mihin linkit osoittavat.



Kuva 12. Objektien yhteydet toisiinsa

Nyt on esitelty Riak-tietokannan rakenne ja tärkeimmät työkalut. Seuraavaksi esitellään lyhyesti toinen tässä työssä käytettävistä tietokannoista. Tämä tietokanta on SQL-tietokanta toisin kuin edellä esitelty Riak. Sen jälkeen kerrotaan yleisesti, mitä ovat suunnittelumallit ja mihin niitä tarvitaan sekä tarkemmin muutamasta tässä työssä käytetystä suunnittelumallista.

3.2 PostgreSQL

PostgreSQL on avoimen lähdekoodin relaatiotietokanta, jota käytetään tässä työssä tietolähteenä Riakin kanssa. PostgreSQL-tietokannoissa käytetään standardoitua IBM:n kehittämää SQL-kyselykieltä, jonka avulla tietokantaan tehdään hakuja, muutoksia ja lisäyksiä. PostgreSQL-tietokantaan ollaan yhteydessä PostgreSQL JDBC -ajurin avulla, joka löytyy valmiiksi tässä työssä käytetystä kehitysympäristöstä, josta kerrotaan tarkemmin luvussa tekninen määrittely.

3.3 Suunnittelumallit

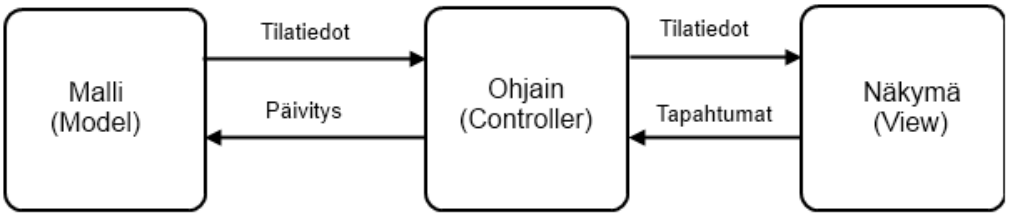
Suunnittelumalli on käytännön kokemuksen pohjalta laadittu ja hyväksi koettu ratkaisu oliopohjaisen ohjelman suunnittelussa esiintyvään tilanteeseen. Suunnittelumallin käytöstä on paljon kokemusta. Se ei ole kerran kokeiltu toimiva ratkaisu vaan se on havaittu useaan otteeseen olevan hyvä ja sopiva ratkaisu tiettyyn tilanteeseen. Suunnittelumallin kolme tärkeintä osaa ovat ongelma, ongelmayhteys ja ratkaisu. Ongelma on se

tilanne, jonka ratkaisemiseen tarvitaan suunnittelumallia. Ongelmayhteys kertoo suunnittelumallin yhteyden eli sen, milloin ja missä sitä voidaan käyttää. Ratkaisu kuvaa ongelman ratkaisuun tarvittavan ohjelmointirakenteen, sen osat (luokat, oliot) ja osien väliset suhteet sekä niiden vuorovaikutukset (funktiokutsut) yleisellä tasolla.

Suunnittelumallit luokitellaan usein sen tarkoituksen (luonti, rakenne, käyttäytyminen) ja kohteen (olio, luokka) perusteella. Mallit voidaan myös luokitella sen mukaan, mitä malleja käytetään yhdessä. Tässä työssä on käytetty esimerkiksi MVC-mallia (Model-View-Controller pattern) ja luontimallia tehdasfunktio (Factory Method), joista kerrotaan tarkemmin seuraavaksi.

3.3.1 MVC-malli (Model-View-Controller pattern)

MVC-malli on suunnittelumalli, jossa ohjelman käyttöliittymä ja varsinainen toiminnallisuus eli sovelluslogiikka halutaan erottaa toisistaan, jolloin päästään tilanteeseen, jossa yhden osan muuttuminen ei vaikuta kovinkaan suuresti toiseen osaan. MVC-malli koostuu kolmesta osasta: mallista (model), näkymästä (view) ja ohjaimesta (controller), joista suunnittelumalli saa nimensä.



Kuva 13. MVC-mallin osat ja niiden väliset suhteet [4].

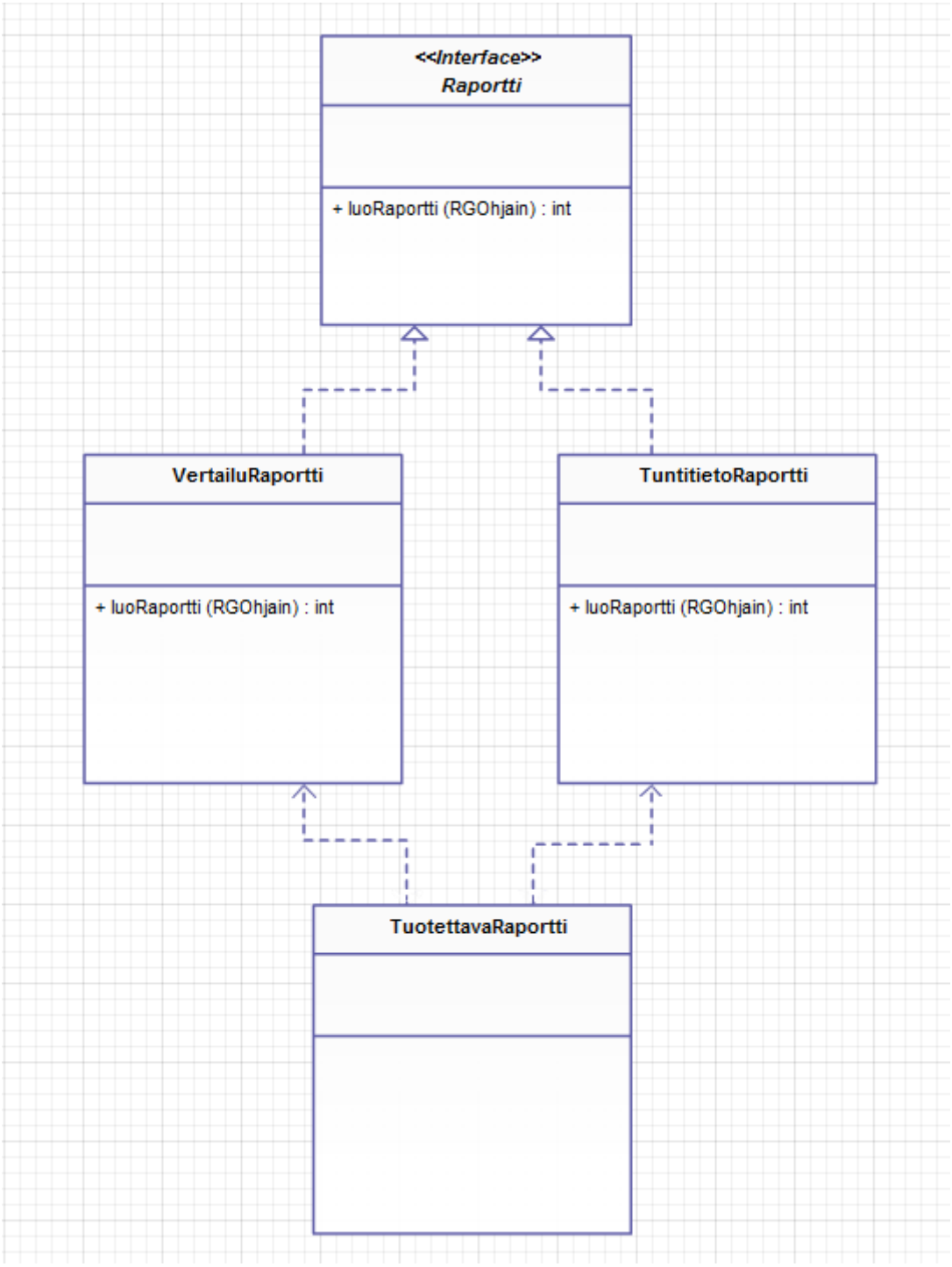
Yllä olevassa kuvassa 13 on esitetty osien välinen vuorovaikutus. Käyttäjä näkee vain näkymä-osan, joka keskustelee ohjain-osan kanssa. Kun käyttäjä tekee toiminnon, niin näkymä kertoo ohjaimelle, mitä käyttäjä tekee. Sen jälkeen ohjain kertoo malli-osalle, mitä toiminnallisuutta tulee käyttää. Kun malli on toimintonsa suorittanut, se välittää tuloksen takaisin ohjaimelle. Ohjain siirtää tiedon takaisin näkymä-osalle, jolloin käyttäjälle näkyy tehdyn toiminnon tulos.

3.3.2 Luontimallit

Näissä suunnittelumalleissa ohjelman olion rakentaminen esitetään käsitteellisellä tasolla. Luontimallit kuvaavat sen kuinka, koska ja mitä olioita luodaan eli olioiden luomisprosessin. Luontimallit piilottavat konkreettisesti käytettävien tuotantoluokkien näkyvyyden niitä käyttäviltä luokilta eli tuotantoluokkia käyttävä ohjelma tuntee niistä vain niiden abstraktissa luokassa esitettävän rajapinnan. Tämä mahdollistaa tuotantoluokkien ja niiden välisten suhteiden muokkaamisen ilman, että se vaikuttaisi luokkia tilaavan ohjelman toimintaan.

3.3.3 Tehdasfunktio (Factory Method)

Tehdasfunktio-suunnittelumallissa määritellään rajapinta, joka määrittää kehykset eli yhteiset ominaisuudet rajapintaa toteuttaville luokille, mutta ei välttämättä ota kantaa itse ominaisuuden toteutukseen vaan sysää sen rajapintaa toteuttaville luokille. Seuraavassa kuvassa 14 esitellään tehdasfunktion rakennetta luokkakaavion muodossa. Kuvassa on käytetty raporttigeneraattorin luokkia.



Kuva 14. Tehdasfunktio esitettynä luokkakaavion muodossa.

Kuvan yläosassa oleva Raportti-rajapinta määrittää, mitä ominaisuuksia sitä toteuttavilla raporteilla pitää olla. Tässä tapauksessa on vain yksi ominaisuus, ja se on luoRaportti()-metodi. Kyseiselle metodille ei ole kirjoitettu toteutusta, vaan toteutuksen kirjoitus on jätetty Raportti-rajapintaa toteuttaville luokille.

Raporttigeneraattorissa tuotetaan kahdenlaisia raportteja, joten tarvitaan kaksi erilaista raporttia, jotka toteuttavat Raportti-rajapintaa. Nämä kaksi luokkaa ovat kuvassa Raportti-luokan alapuolella. Ne sisältävät luoRaportti()-metodin sekä sen toteutuksen. Kun käyttäjä tuottaa raportin, jota kuvassa edustaa TuotettavaRaportti-luokka, niin käyttäjä valitsee, mitä Raportti-luokan toteutusta käytetään.

4 Toiminnallinen määrittely

Tässä luvussa kuvataan, mitä toimintoja raporttigeneraattorilla voi tehdä ja miten käyttäjä voi niitä tehdä sekä mitä vaatimuksia raporttigeneraattorille on asetettu. Toteutukseen liittyvät yksityiskohdat ja tekniset toteutukset kerrotaan teknisessä määrittelyssä.

4.1 Vaatimukset

Raporttigeneraattorin suunnittelussa lähdettiin liikkeelle perehtymällä Energiateollisuus ry:n asettamiin formaatteihin, joita raporttigeneraattorin pitää noudattaa. Näissä formaateissa määritettiin muun muassa raporttien tiedostomuoto, tiedostonimi, sekä sisältö. Alla on avattu hieman näitä formaatteja, jotta ohjelman toiminnallisuus ja rakenne olisi helpompi ymmärtää. Tarkemmat ja viralliset formaattimäärittelyt sekä malliraportit ovat liitteenä.

Formaattien mukaan tasevirheiden tuntitiedot -raportin tiedostomuoto tulee olla CSV, koska useimmat tietojärjestelmät, joihin raportin tietoja tallennetaan pystyvät lukemaan koneellisesti CSV-tiedostoja. CSV-tiedostojen käsittely onnistuu myös yleisimmillä taulukkolaskentaohjelmilla kuten esimerkiksi Microsoft Excelillä. CSV on tiedostomuoto, jossa tallennetaan pilkuilla eroteltua taulukkomuotoista tietoa tekstitiedostoon. Raportin tiedostonimen pitää alkaa aina ”Tasevirhetuntitiedot”-sanalla, jonka jälkeen tulee alaviivoilla eroteltuina lähettäjän osapuolitunnus, vastaanottajan osapuolitunnus, aikaväli, jota raportti käsittelee sekä sekvenssinumero (esim. Tasevirhetuntitiedot_JVH000_MYYJ_YYYYMMDDHHMMZ_YYYYMMDDHHMMZ_1.csv). Sekvenssinumero on juokseva luku, joka tekee tiedostosta yksilöllisen.

Tasevirhe tuntitiedot -raportin tulee olla taulukko, jonka ensimmäiselle riville tulee otsikkotiedot ensimmäistä solua lukuun ottamatta, johon tulee verkonhaltijan ja myyjän

osapuolitunnukset. Toiselle riville tulee käyttöpaikkanumero, jota sarakkeessa oleva aikasarja koskee, lukuun ottamatta ensimmäistä solua, johon tulee teksti ”Aikaleima / KP”. Se tarkoittaa, että solun alapuolelle tulee mittausten aikaleimat ja solun vasemmalle puolelle tulevat käyttöpaikkanumerot. Yhtä käyttöpaikkaa kohden on kolme eri tietoa vierekkäisissä sarakkeissa, joten sama käyttöpaikkanumero on kaikissa kolmessa sarakkeessa. Käyttöpaikat ovat suuruusjärjestyksessä pienimmästä suurimpaan. Aikaleimat kattavat tunnin mittaisen ajan ja ne on alkuaikaleimattu (esimerkiksi 13:00-14:00 merkataan 13:00). Aikaleimat ovat UTC-0-aikaa, mutta ajoittuvat Suomen virallisen ajan mukaiseen vuorokauden vaihteeseen. Vuorokausi alkaa siis joko 22:00 tai 21:00 riippuen onko talvi- vai kesäaika. Aikaleimat esitetään muodossa: ”2013-01-31T22:00:00Z”.

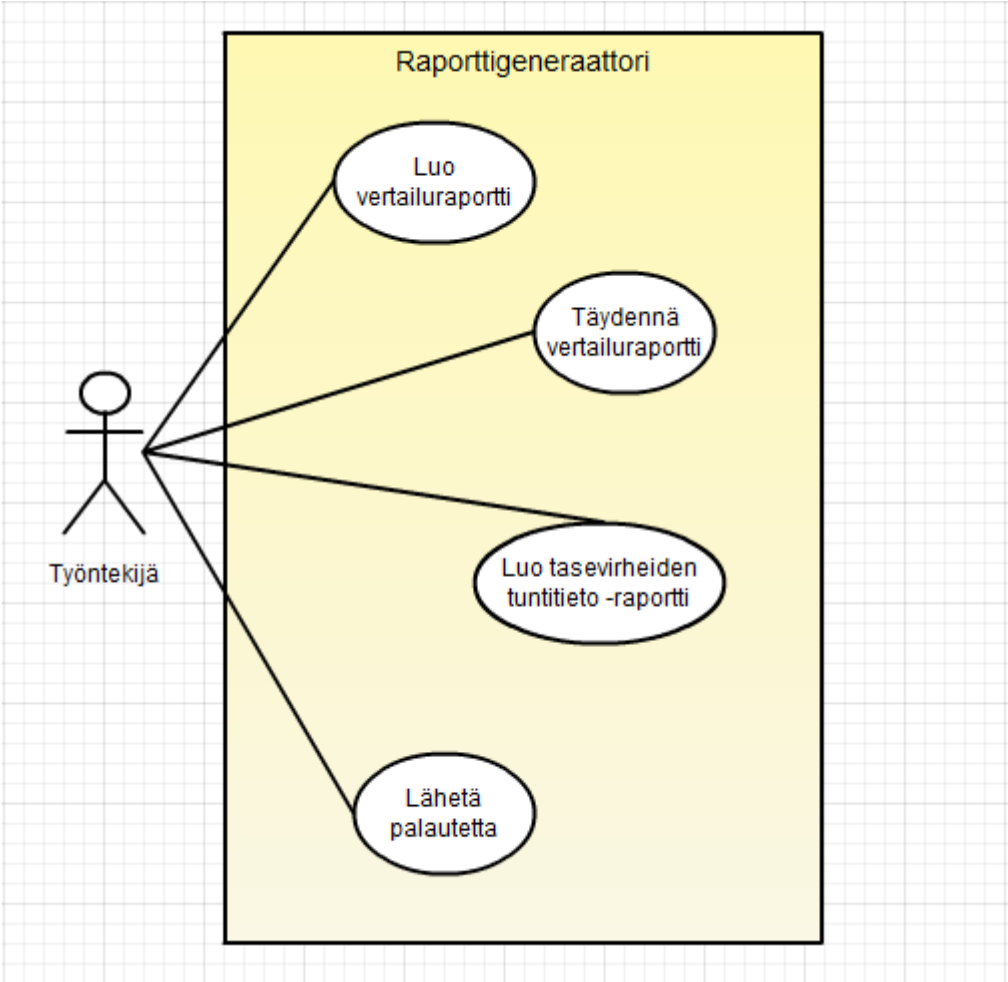
Kun tasevirheiden tuntitiedot -raportin tulee olla CSV-muotoinen, niin vertailuraportin tulee olla taas xls-muotoinen eli Microsoft Excel -tiedosto. Raportin tiedostonimen pitää alkaa ”Tasevertailu”-sanalla, jonka jälkeen tulee alaviivoilla eroteltuina verkonhaltijan osapuolitunnus ja myyjän osapuolitunnus (esim. Tasevertailu_JVH000_MYYJ.xls). Raportti sisältää kaksi välilehteä. Toiselle välilehdelle tulee myyjän ja toiselle verkonhaltijan tiedot. Kukin osapuoli täydentää itse välilehtensä oman järjestelmänsä tiedoilla. Sarakkeeseen A tulee kyseisen myyjän tuntimittauksen piirissä olevat käyttöpaikat tunnuksineen kasvavassa järjestyksessä. Sarakkeeseen B tulee käyttöpaikan myyjätiedon voimaantulopäivämäärä ja sarakkeeseen C tulee käyttöpaikan myyjätiedon päättymispäivämäärä. Sarakkeeseen D lasketaan vertailtavan ajanjakson käyttöpaikkakohtainen yhteenlaskettu sähköenergian käyttö. Sarakkeisiin E, F, G ja H tulee käyttöpaikan tietojen poikkeamat toisen osapuolen tietoihin verrattuna. Esimerkiksi sarakkeeseen H tulee oman sekä toisen osapuolen yhteenlasketun sähköenergian käytön erotus. Nämä sarakkeet kertovat nopeasti, onko oman ja toisen osapuolen tiedoissa eroja.

Formaattien lisäksi Energiateollisuus ry on asettanut suositustakarajan, jolloin yritysten tulisi olla valmiita tuottamaan yllä esiteltyjä tasevirheen korjausraportteja. Takaraja oli 15.3.2013.

4.2 Toiminnot

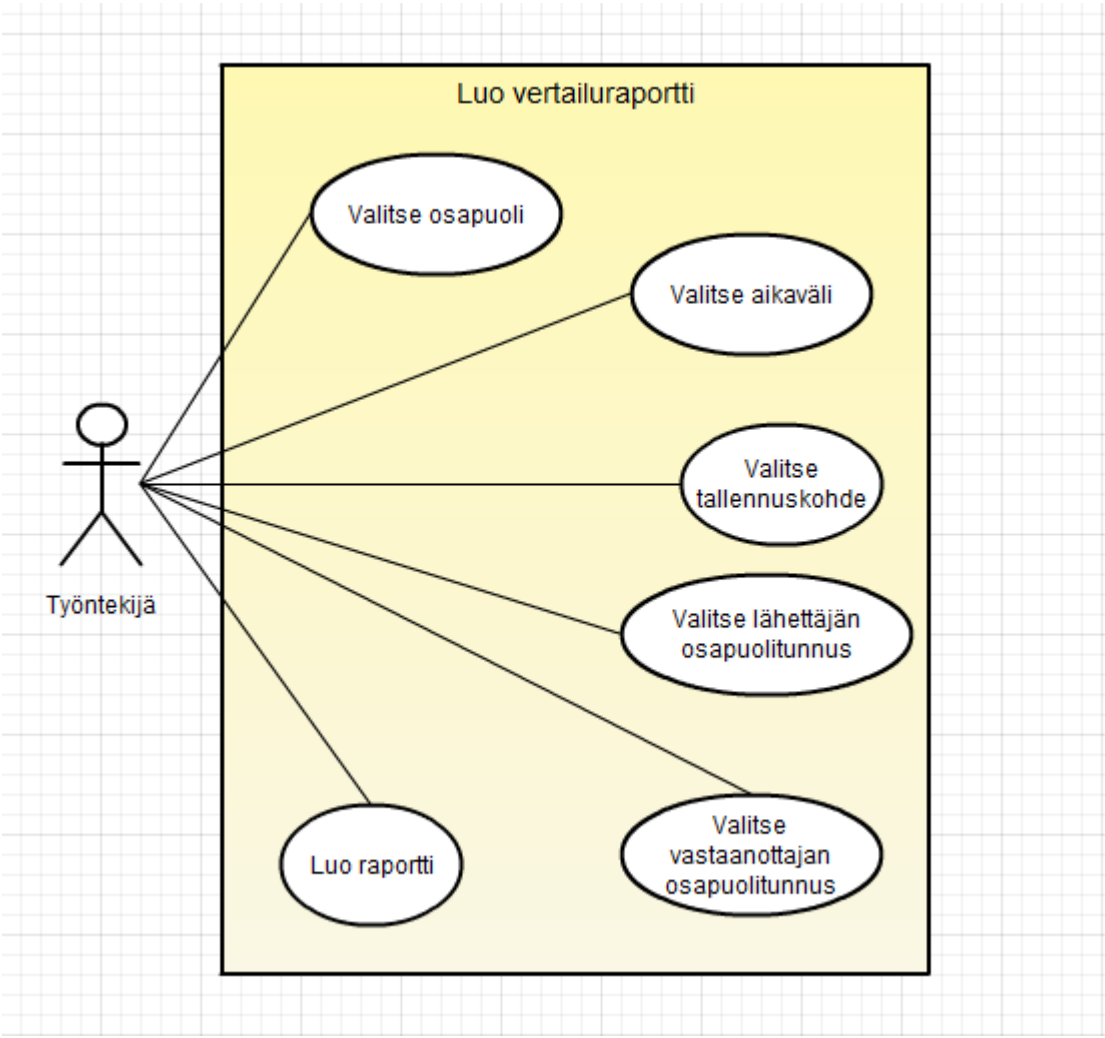
Sovelluksen toiminnot voidaan jakaa neljään suurempaan toimintokokonaisuuteen: vertailuraportin luonti, vertailuraportin täydennys, tasevirheiden tuntitieto -raportin luonti

ja palautteen lähettäminen, jotka on esitetty alla olevassa kuvassa 15. Nämä neljä toimintokokonaisuutta on pilkottu myös pienemmiksi osiksi, joista kerrotaan myöhemmin. Pilkkominen auttaa käyttäjää ymmärtämään mitä toimintoja hän tulee tekemään ohjelmaa käyttäessään. Seuraavaksi käydään tarkemmin läpi nämä kuvassa 15 esitetyt neljä kokonaisuutta.



Kuva 15. Raporttigeneraattorin käyttötapauskaavio.

Vertailuraportin käsittelyn osa ”Luo vertailuraportti” on toimintokokonaisuus, jossa työntekijä valitsee oman osapuolensa, aikavälin, raportin tallennuskohteen sekä vastaanottajan ja lähettäjän osapuolitunnukset. Valittuaan kaikki tiedot käyttäjä voi luoda vertailuraportin. Raporttia ei siis voi luoda ennen kuin kaikki muut toiminnot on tehty. Vertailuraportin luonnin käyttötapauskaavio on esitetty kuvassa 16.



Kuva 16. Vertailuraportin luomisen käyttötapauskaavio.

Jokaisesta käyttötapauksesta on annettu tarkemmat kuvaukset seuraavissa taulukoissa, joissa kerrotaan tekijä, lyhyt kuvaus, esitiedot ja lopputulos. Taulukossa 1 esitetään käyttötapauksen ”Valitse osapuoli”-kuvaus. Käyttötapauksessa käyttäjä, eli tämän työn kohdalla Rejlers Oy:n työntekijä, valitsee täydennetäänkö VERKKO- vai MYYJÄ-välilehden tiedot, jonka jälkeen raporttigeneraattori tietää kummalle raportin välilehdelle tiedot pitää tallentaa.

Taulukko 1. Käyttötapaus ”Valitse osapuoli”

Käyttötapaus	KT1 Valitse osapuoli
Tekijä	Työntekijä

Kuvaus	Tekijä valitsee osapuolen, joka määrää mille raportin välilehdelle tiedot tallennetaan.
Esitiedot	Ohjelma on avattu ja valittu välilehti ”Luo vertailuraportti”.
Lopputulos	Tiedetään kummalle raportin välilehdelle tiedot pitää tallentaa.

Käyttötapauksessa ”valitse aikaväli” työntekijä valitsee aikavälin, jolta vertailuraportin tietoja vertaillaan. Tämän jälkeen raporttigeneraattori tietää aikavälin, jolta tiedot haetaan (taulukko 2).

Taulukko 2. Käyttötapaus ”Valitse aikaväli”

Käyttötapaus	KT2 Valitse aikaväli
Tekijä	Työntekijä
Kuvaus	Tekijä valitsee aikavälin, jonka ajalta tietoja halutaan vertailla.
Esitiedot	Ohjelma on avattu ja valittu välilehti ”Luo vertailuraportti”.
Lopputulos	Tiedetään miltä aikaväliltä tiedot pitää hakea.

Käyttötapauksessa ”Valitse tallennuskohde” työntekijä valitsee raportille tallennuskohteen, jonka jälkeen raporttigeneraattori tietää mihin raportti tallennetaan (taulukko 3).

Taulukko 3. Käyttötapaus ”Valitse tallennuskohde”

Käyttötapaus	KT3 Valitse tallennuskohde
Tekijä	Työntekijä
Kuvaus	Tekijä valitsee tallennuskohteen, johon raportti halutaan tallentaa.

Esitiedot	Ohjelma on avattu ja valittu välilehti ”Luo vertailuraportti”.
Lopputulos	Tiedetään mihin raportti pitää tallentaa.

Käyttötapauksessa ”Valitse lähettäjän osapuolitunnus” (taulukko 4) työntekijä valitsee raportin lähettäjän osapuolitunnuksen. Valinnan jälkeen raporttigeneraattori tietää lähettäjän osapuolitunnuksen, jota tarvitaan osapuolten tunnistamisen lisäksi myös tiedoston nimeämiseen.

Taulukko 4. Käyttötapaus ”Valitse lähettäjän osapuolitunnus”

Käyttötapaus	KT4 Valitse lähettäjän osapuolitunnus
Tekijä	Työntekijä
Kuvaus	Tekijä valitsee lähettäjän osapuolitunnuksen, joka tulee osaksi tiedostonimeä.
Esitiedot	Ohjelma on avattu ja valittu välilehti ”Luo vertailuraportti”.
Lopputulos	Tiedetään lähettäjän osapuolitunnus, joka on tiedostonimen toinen tarvittava osa.

Käyttötapauksessa ”Valitse vastaanottajan osapuolitunnus” (taulukko 5) työntekijä valitsee raportin vastaanottajan osapuolitunnuksen. Valinnan jälkeen raporttigeneraattori tietää vastaanottajan osapuolitunnuksen, jota tarvitaan yhtälailla osapuolten tunnistamisessa ja tiedoston nimeämisessä kuin lähettäjän osapuolitunnusta.

Taulukko 5. Käyttötapaus ”Valitse vastaanottajan osapuolitunnus”

Käyttötapaus	KT5 Valitse vastaanottajan osapuolitunnus
Tekijä	Työntekijä

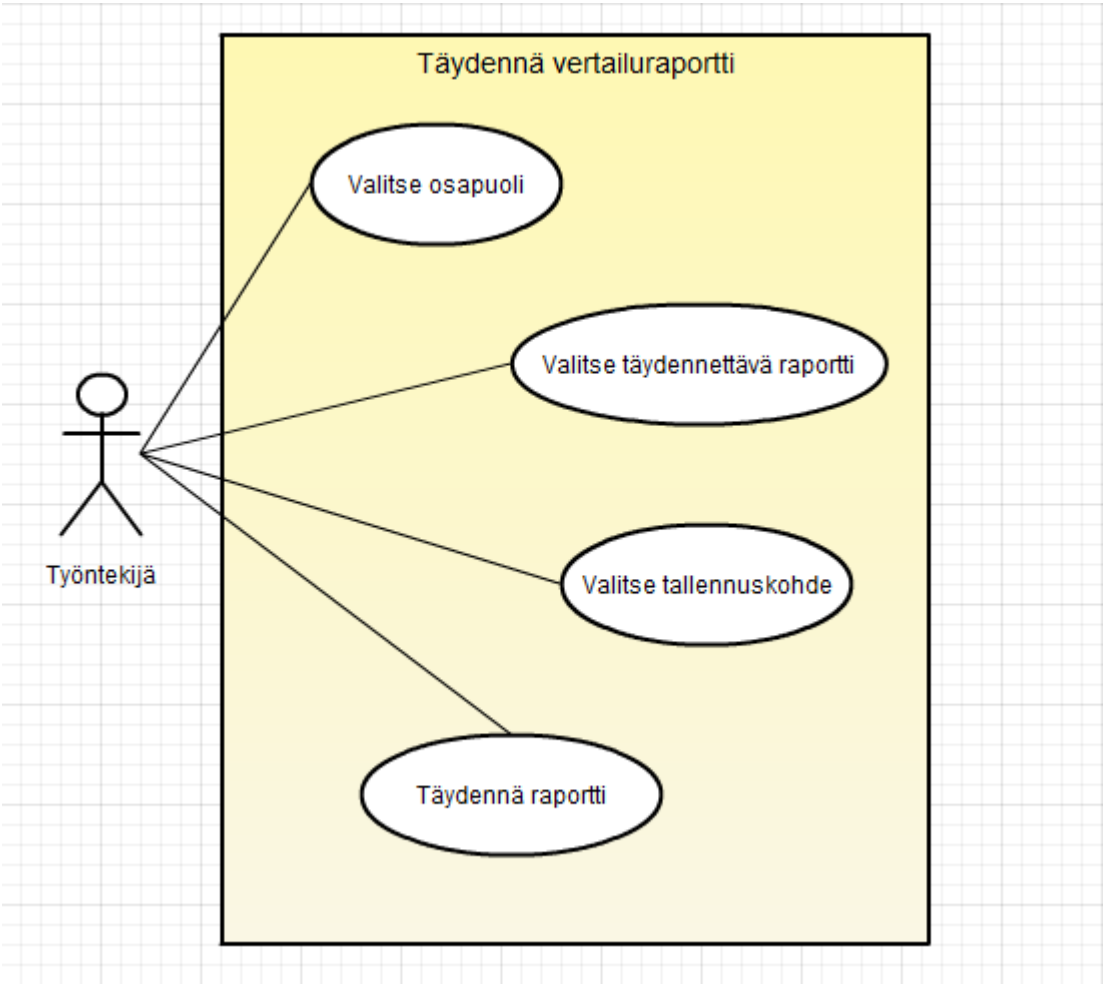
Kuvaus	Tekijä valitsee vastaanottajan osapuolitunnuksen, joka tulee osaksi tiedostonimeä.
Esitiedot	Ohjelma on avattu ja valittu välilehti ”Luo vertailuraportti”.
Lopputulos	Tiedetään vastaanottaja osapuolitunnus, joka on tiedostonimen toinen tarvittava osa.

Kun kaikki viisi tähän mennessä käsiteltyä käyttötapausta on suoritettu, niin raporttigeneraattorilla on kaikki tarpeellinen tieto raportin luomista varten ja työntekijä voi siirtyä käyttötapaukseen ”Luo raportti”, jossa työntekijä luo uuden raportin. Käyttötapauksen lopputuloksena saadaan valmis raportti, joka on täydennetty halutuilla tiedoilla. Käyttötapausta ”Luo raportti” ei siis voi suorittaa ennen kuin kaikki viisi aikaisempaa käyttötapausta on suoritettu.

Taulukko 6. Käyttötapaus ”Luo raportti”

Käyttötapaus	KT6 Luo raportti
Tekijä	Työntekijä
Kuvaus	Työntekijä käynnistää raportin luonnin
Esitiedot	Ohjelma on avattu ja valittu välilehti ”Luo vertailuraportti” sekä käyttötapaukset KT1, KT2, KT3, KT4, ja KT5 on suoritettu.
Lopputulos	Valmis raportti, joka on täydennetty tarvittavilla tiedoilla.

Vertailuraportin luominen on vain yksi osa vertailuraporttien käsittelyä. Vertailuraporttien käsittelyn toinen osa on vertailuraportin täydentäminen, jonka käyttötapauskaavio on esitetty kuvassa 17.

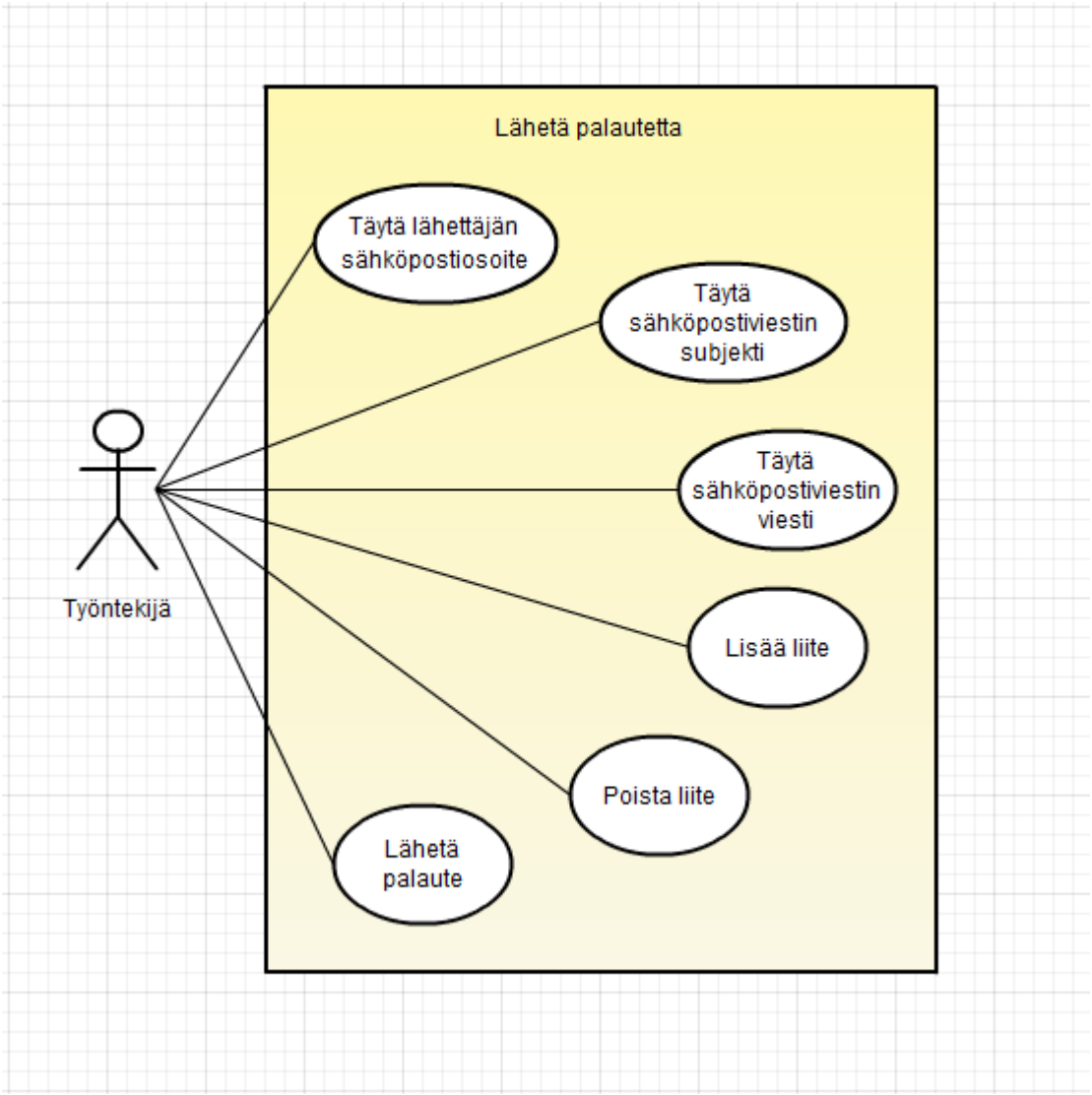


Kuva 17. Vertailuraportin täydennyksen käyttötapauskaavio

Vertailuraportin täydennyksessä työntekijä täydentää jo olemassa olevaa vertailuraporttia, jonka toinen osapuoli on täyttänyt omalta osaltaan. Täydennys eroaa luomisesta sen verran, että osapuolitunnuksia ja aikaväliä ei tarvitse valita, koska ne saadaan täydennettävästä raportista. Täydennettävään raporttiin tulee yksi uusi käyttötapaus, jossa valitaan täydennettävä raportti. Käyttötapauksen tuloksena ohjelma tuntee tiedoston, josta saadaan osapuolitunnukset sekä aikaväli. Tähän tiedostoon työntekijä täydentää tietonsa. Käyttötapauksen suorittamiseen ei tarvitse olla kuin ohjelma avattuna ja Täydennä vertailuraportti -välilehti valittuna.

Kolmas raporttien tuottamiseen liittyvä toimintokokonaisuus on ”Luo tasevirheiden tuntitiedot -raportti”. Siinä on lähes samat käyttötapaukset kuin Luo vertailuraportti -toimintokokonaisuudessa. Ainoa ero on se, että tasevirheiden tuntitietoraportin luonnissa ei ole käyttötapausta ”Valitse osapuoli”.

Palautteen lähettäminen on ainoa toimintokokonaisuus, joka ei liity raporttien tuottamiseen. Seuraavassa kuvassa 18 on palautteen lähettämisen käyttötapauskaavio.



Kuva 18. Palautteen lähettämisen käyttötapauskaavio

Täytä lähettäjän sähköpostiosoite -käyttötapauksessa käyttäjä syöttää tekstikenttään oman sähköpostiosoitteensa. Ohjelma ei tarkasta, onko kenttää syötetty teksti sähköpostiosoite, mutta ohjelma vaatii, että kentässä on tekstiä ennen kuin palautteen voi lähettää. Jos kenttä on tyhjä, niin ohjelma ilmoittaa, että tarvittavat kentät pitää täyttää. Täytä sähköpostiviestin subjekti -käyttötapauksessa käyttäjä syöttää tekstikenttään haluamansa sähköpostiviestin subjektin. Ohjelma tarkastaa, että kenttä ei ole tyhjä, kun palautetta lähetetään. Jos kenttä on tyhjä, niin ohjelma ei lähetä palautetta, vaan ilmoittaa, että pitää täydentää tarvittavat kentät. Täytä sähköpostiviestin viesti -

käyttötapauksessa käyttäjä kirjoittaa palautteen tekstikenttään, jonka voi jättää tyhjäksi, jos esimerkiksi subjekti kertoo kaiken tarvittavan tiedon palautteesta.

Lisää liite -käyttötapauksessa käyttäjä valitsee palautteeseen lisättävän liitteen. Käyttäjä voi liittää palautteeseen vain yhden liitteen. Ohjelma antaa myös käyttäjän lisätä vain tiedostoja liitteeksi. Esimerkiksi kansion lisääminen liitteeksi ei onnistu. Poista liite -käyttötapauksessa käyttäjä poistaa lisätyn liitteen.

Lähetä palaute -käyttötapauksessa käyttäjä napsauttaa lähetä-painiketta, jolloin ohjelma tarkistaa, onko tarvittavat kentät täytetty. Jos tarvittavat kentät on täytetty, niin ohjelma lähettää viestin, jossa lähettäjänä, subjektina ja viestinä ovat käyttäjän syöttämät tiedot. Muutoin ohjelma ilmoittaa, ettei tarvittavia kenttiä ole täytetty.

5 Tekninen määrittely

Tekninen määrittely esittää ratkaisut toiminnallisen määrittelyn esittämiin toimintoihin. Ensin esitellään sovelluksen ulkoinen rajapinta, ohjelmointikieli ja ohjelmointiympäristö. Sitten kuvataan sidosryhmien ja luokkakaavion avulla ohjelman arkkitehtuuria ja lopuksi kerrotaan tärkeimmistä kirjastoista, joita tarvitaan ohjelman toteuttamiseen.

5.1 Smac API -rajapinta

Smac API -rajapinta on Rejlers Oy:n toimesta kehitteillä oleva rajapinta Riak-tietokannan käsittelyä varten. Rajapinta aiheuttaa sen, että raporttigeneraattori ei ole suoraan yhteydessä Riak-tietokantaan. Kaikki raporttigeneraattorin ja Riak-tietokannan välinen kommunikointi tapahtuu Smac API -rajapinnan kautta. Tämä kommunikointi Smac API -rajapinnan kanssa suoritetaan rivinvaihdoilla erotetuilla JSON-objekteilla käyttäen mitä tahansa http:tä osaavaa clientia eli asiakasohjelmaa. Smac API -rajapinta on suunniteltu helpottamaan kommunikointia Riak-tietokannan kanssa. Se ei kuitenkaan ole pakollinen. Raporttigeneraattori olisi mahdollista toteuttaa myös ilman Smac API -rajapintaa.

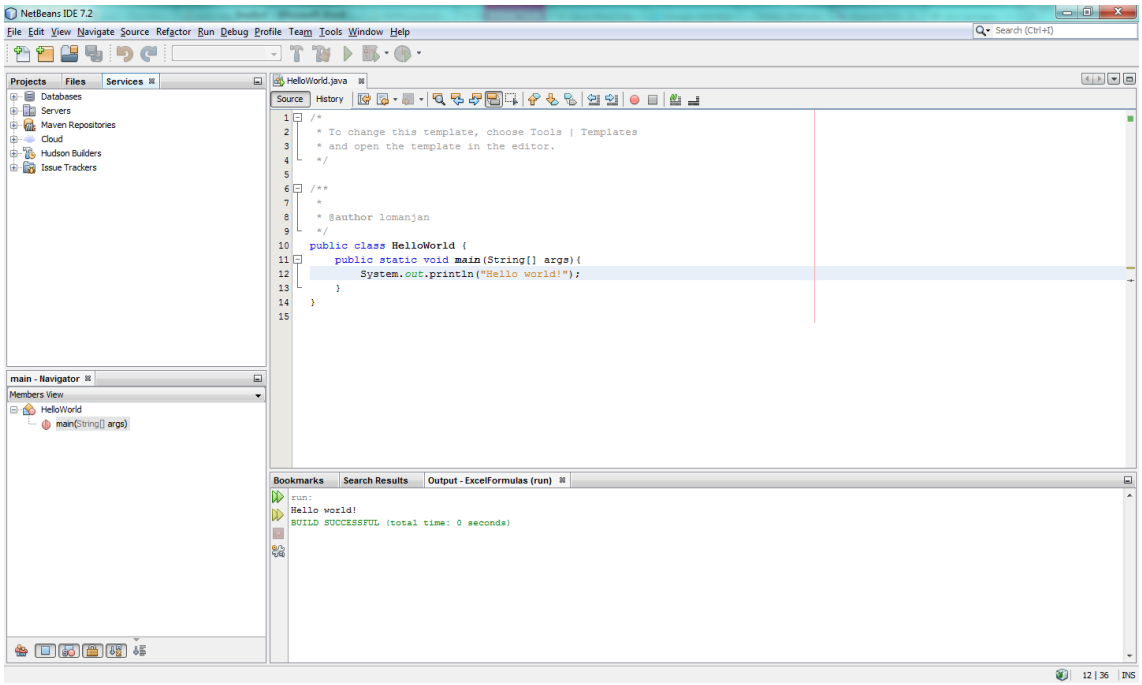
5.2 Ohjelmointikieli

Ohjelma on toteutettu käyttäen Java-ohjelmointikieltä. Java on Sun Microsystemsin kehittämä ohjelmistoalusta. Se sisältää laitteistoriippumattoman oliopohjaisen ohjelmointikielen, ajoympäristön, virtuaalikoneen ja luokkakirjastot. Java on julkaistu vapaana ohjelmistona GNU GPL -lisenssillä.

Java-ohjelmointikielestä julkaistaan eri versiota kehitysympäristön (JDK) kehittymisen myötä. Tässä työssä on käytetty JDK 1.6:sta. Kehitysympäristö pitää sisällään kääntäjän (javac), kehitystyökalut (jar, javadoc, jdb) ja ajoympäristön (JRE), joka tarvitaan käännettyjen ohjelmien ajamiseen. Ajoympäristö sisältää virtuaalikoneen (JVM), joka sisältää ajonaikaisen käännöksen konekielelle. Ajoympäristöön kuuluu myös luokkakirjastot.

5.3 Ohjelmointiympäristö

Raporttigeneraattorin ohjelmointiympäristönä on käytetty NetBeansin versiota 7.2. NetBeans (kuva 19) on avoimen lähdekoodin integroitu ohjelmistoympäristö Java-, JavaScript-, PHP-, Python-, Ruby-, Groovy-, C-, C++-, Scala ja Clojure-ohjelmointikielille. Netbeansin pelkkä käyttäminen vaatii Javan ajoympäristön (JRE). Jotta Netbeans-ohjelmointiympäristöllä pystyisi kehittämään Java-ohjelmia, se vaatii lisäksi Java-kehitysympäristön (JDK).



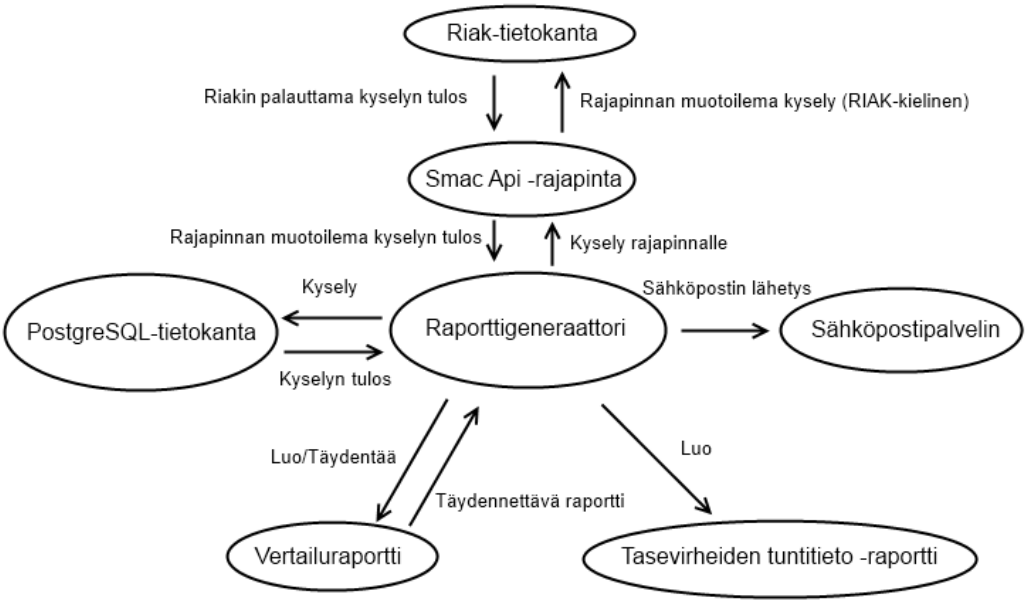
Kuva 19. NetBeans IDE 7.2-ohjelmointiympäristö

Kuvassa 19 on perusnäkymä ohjelmointiympäristöstä. Itse ohjelman koodi kirjoitetaan kuvassa isoimpaan oikeassa yläkulmassa olevaan valkoiseen ikkunaan. Vasemmassa yläkulmassa olevasta ikkunasta voi nähdä projektin tiedostot ja rakenteen. Oikeassa alareunassa olevasta ikkunasta nähdään ohjelman tuottamat näytölle tulevat tulostukset.

5.4 Arkkitehtuuri

5.4.1 Ulkoiset komponentit

Raporttigeneraattorin toiminnallisuus koostuu monesta osasta. Seuraavassa kuvassa 20 on esitelty generaattorin ulkoiset komponentit ja niiden kytkökset toisiinsa.



Kuva 20. Raporttigeneraattorin ulkoiset komponentit

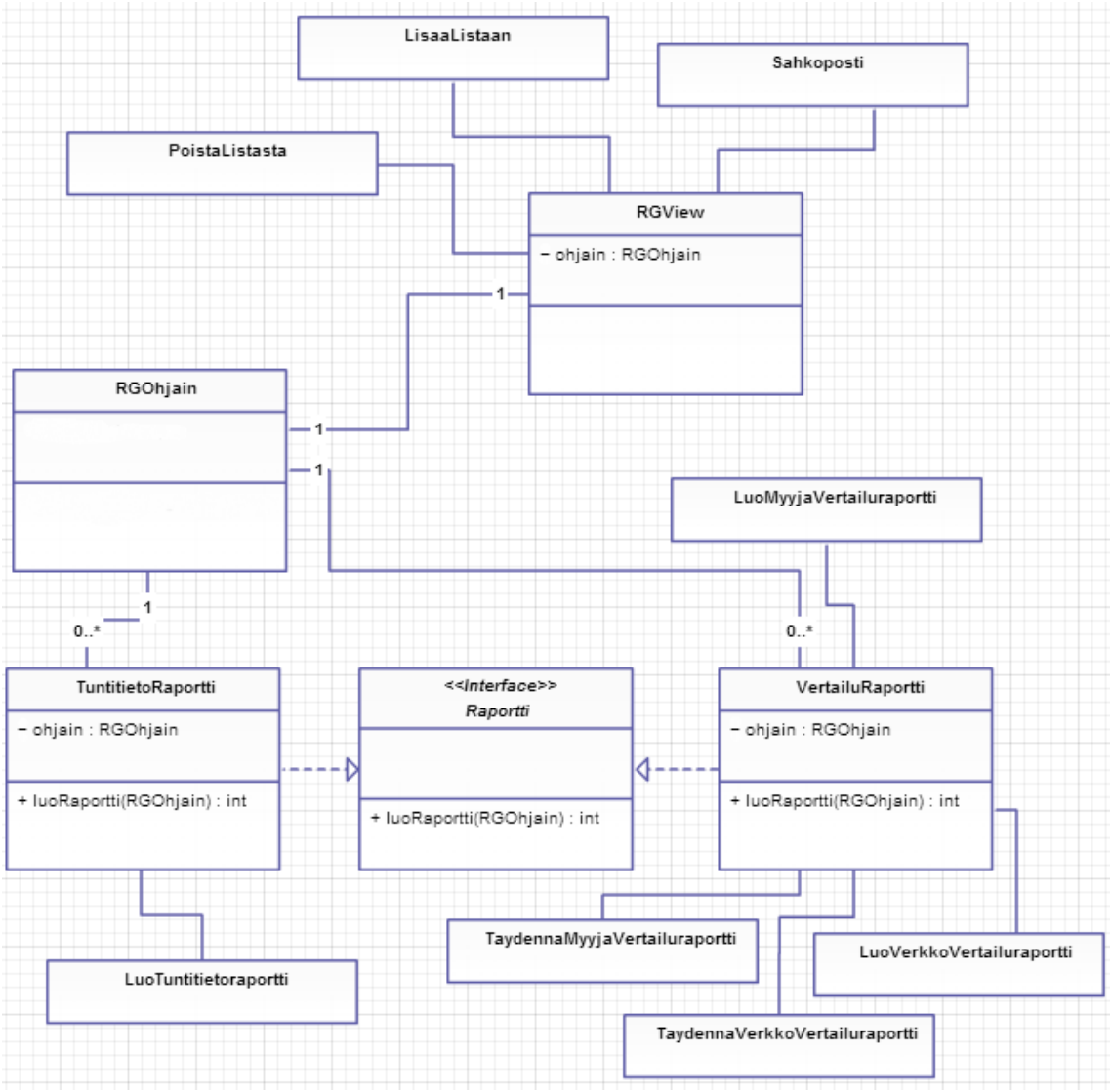
Kuten kuvasta 20 voi nähdä, niin raporttigeneraattori ei ole suoraan yhteydessä Riak-tietokantaan vaan se on yhteydessä Smac API -rajapintaan, joka on yhteydessä Riak-tietokantaan, josta haetaan taseikkunan sulkeutumisen jälkeen muuttuneita mittaustietoja. Raporttigeneraattori on yhteydessä myös PostgreSQL-tietokantaan, josta haetaan tiedot, joita tarvitaan Riak-tietokantaan kohdistuvien kyselyiden ja generaattorin tuottamien raporttien muodostamisessa.

Generaattori tuottaa vertailuraportteja ja tasevirheiden tuntitieto -raportteja sekä täydentää olemassa olevaa vertailuraporttia, jonka toinen osapuoli on lähettänyt. Lisäksi generaattori tarvitsee sähköpostipalvelinta, mikäli ohjelman käyttäjä haluaa lähettää palautetta generaattorista.

5.4.2 Luokkakaavio

Luokkakaavio on mallintamisen apuväline, jolla kuvataan olio-ohjelman luokkia, luokkien ominaisuuksia sekä luokkien välisiä assosiaatioita eli suhteita. Luokkakaavion pe-

rusteella ohjelmointiympäristö voi osata tuottaa ohjelmakoodia, jossa on samat ominaisuudet ja suhteet kuin itse luokkakaaviossa. Tämä nopeuttaa varsinkin suuren ohjelman ohjelmointia, koska luokat ja suurin osa metodeista sekä metodien parametreista luodaan valmiiksi. Luokkakaavio helpottaa myös ohjelman rakenteen ja luokkien assosiaatioiden hahmottamista. Seuraavassa kuvassa 21 on esitetty raporttigeneraattorin luokkakaavio. Kuvassa näkyy vain luokkien tärkeimmät metodit, jotta kuvan ymmärtäminen ja raporttigeneraattorin rakenteen hahmottaminen ei hankaloituisi suuren tietomäärän takia.



Kuva 21. Luokkakaavio

Kuvasta 21 nähdään, kuinka RGOhjain-luokka liittää ohjelman muut luokat toisiinsa. RGOhjain-luokka edustaa luvussa 3.3 esitellyn MVC-mallin ohjainta, joka välittää tieto-

ja näytön (RGView-luokan) sekä raporttiluokkien välillä. Kuvasta nähdään myös, että VertailuRaportti- ja TuntitietoRaportti-luokat toteuttavat Raportti-rajapinnan, jolloin molemmilla luokilla pitää olla toteutus Raportti-rajapinnan luoRaportti()-metodille. Nämä kaksi edellä mainittua asiaa ovat ohjelman rakenteen kannalta merkittävimmät asiat.

5.5 Tärkeimmät kirjastot

Kirjasto on kokoelma aliohjelmia tai luokkia, joita suoritettavat ohjelmat käyttävät apunaan. Kun suoritettava ohjelma käyttää kirjastoja, niin sen ei tarvitse sisältää joka kerta esimerkiksi luokkaa, joka hoitaa kirjoitusoperaatiot tiedostoon, vaan se voi hakea itselleen kirjaston, jossa nämä toiminnot on toteutettu. Kirjaston luokat saa käyttöön kirjoittamalla suoritettavan ohjelman luokan, joka käyttää kirjaston luokkaa; alkuun import ja kirjaston polku. Esimerkiksi import java.util.Date.

Java API tarjoaa hyvin kattavan kirjastokokoelman, josta löytyvät ratkaisut yleisimpiin toteutuksiin, mutta ei kaikkiin. Seuraavaksi on esitelty tässä työssä käytetyistä kirjastoista tärkeimmät.

5.5.1 Apache POI

Työn toteutuksessa on käytetty Apache POI Java-kirjastoja, jotka on kehittänyt Apache Software Foundation. Näiden kirjastojen avulla ohjelma pystyy kirjoittamaan ja lukemaan tiedostoja, jotka ovat Microsoft Offic-formaatissa. Sovelluksessa käytetään Microsoft Office Excel (.xls) -formaattia muodostettaessa raportti, jossa vertaillaan myyjän ja verkonhaltijan tietoja tasevirheiden löytämiseksi. Alla olevassa kuvassa 22 on esitelty sovelluksessa käytetyt Apache POI-kirjastot.

```
10  import org.apache.poi.hssf.usermodel.HSSFCellStyle;
11  import org.apache.poi.hssf.usermodel.HSSFRow;
12  import org.apache.poi.hssf.usermodel.HSSFSheet;
13  import org.apache.poi.hssf.usermodel.HSSFWorkbook;
14  import org.apache.poi.hssf.util.HSSFColor;
15  import org.apache.poi.ss.usermodel.Font;
16  import org.apache.poi.ss.util.CellRangeAddress;
```

Kuva 22. Apache POI -kirjastoja.

Toteutuksessa käytetyt kirjastot pystyy lataamaan Apache Poin internetsivuilta [7] ilmaiseksi. Toteutuksessa käytettiin Apache Poi:n versiota 3.8.

5.5.2 Apache http Client

Apache http Client -kirjastoja käytetään, kun ohjelma on yhteydessä Smac API -rajapintaan. Kirjastojen avulla Smac API -rajapinnasta saadaan haettua raportteihin tarvittavia tietoja.

5.5.3 PostgreSQL JDBC Driver

PostgreSQL JDBC Driver tarjoaa Java-ohjelmille yhteyden PostgreSQL-tietokantaan ja työkalut sen käsittelemiseen. Työssä käytetty ohjelmointiympäristö NetBeans sisältää valmiiksi PostgreSQL JDBC Driverin. Sen saa käyttöön menemällä projektin ominaisuuksiin ja valitsemalla kohdan kirjastot. Tästä kohdasta löytyy Lisää kirjasto -painike, jolloin aukeaa valikko, josta voi valita PostgreSQL JDBC Driverin. Kyseistä kirjastoa käytetään raporttigeneraattorissa, kun haetaan tietoja PostgreSQL-tietokannasta.

5.5.4 JavaMail API

JavaMail API sisältää kirjastot, jotka mahdollistavat alusta- ja protokollariippumattoman sähköpostiviestien muodostamisen, lukemisen ja lähettämisen. Tässä työssä on käytetty JavaMail API:n versiota 1.4.5 ja sitä on käytetty raporttigeneraattorin hallinnan toteutuksessa. JavaMail API:n saa käyttöön lataamalla sen JavaMail API:n kotisivuilta [10] ja liittämällä sen projektin ominaisuuksien kautta itse projektiin.

6 Raporttigeneraattorin toteutus

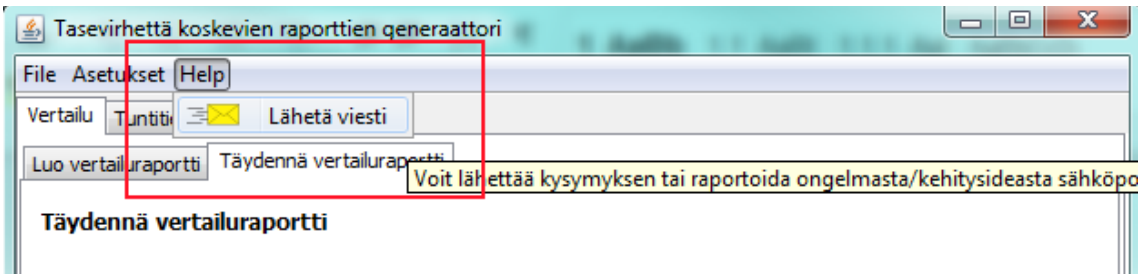
6.1 Raporttigeneraattorin hallinnan toteutus

Raporttigeneraattori ei koostu pelkästään raporttien luontiin ja täydennykseen tarvittavista osista. Generaattorin toimintaa pitää pystyä hallitsemaan, jotta sen käyttö olisi käyttäjäystävällisempää. Hallinta on toteutettu valikkopalkin avulla, joka koostuu kol-

mesta valikosta: Filestä, Asetuksista ja Helpistä. Seuraavaksi käydään läpi, mitä toimintoja näistä valikoista löytyy.

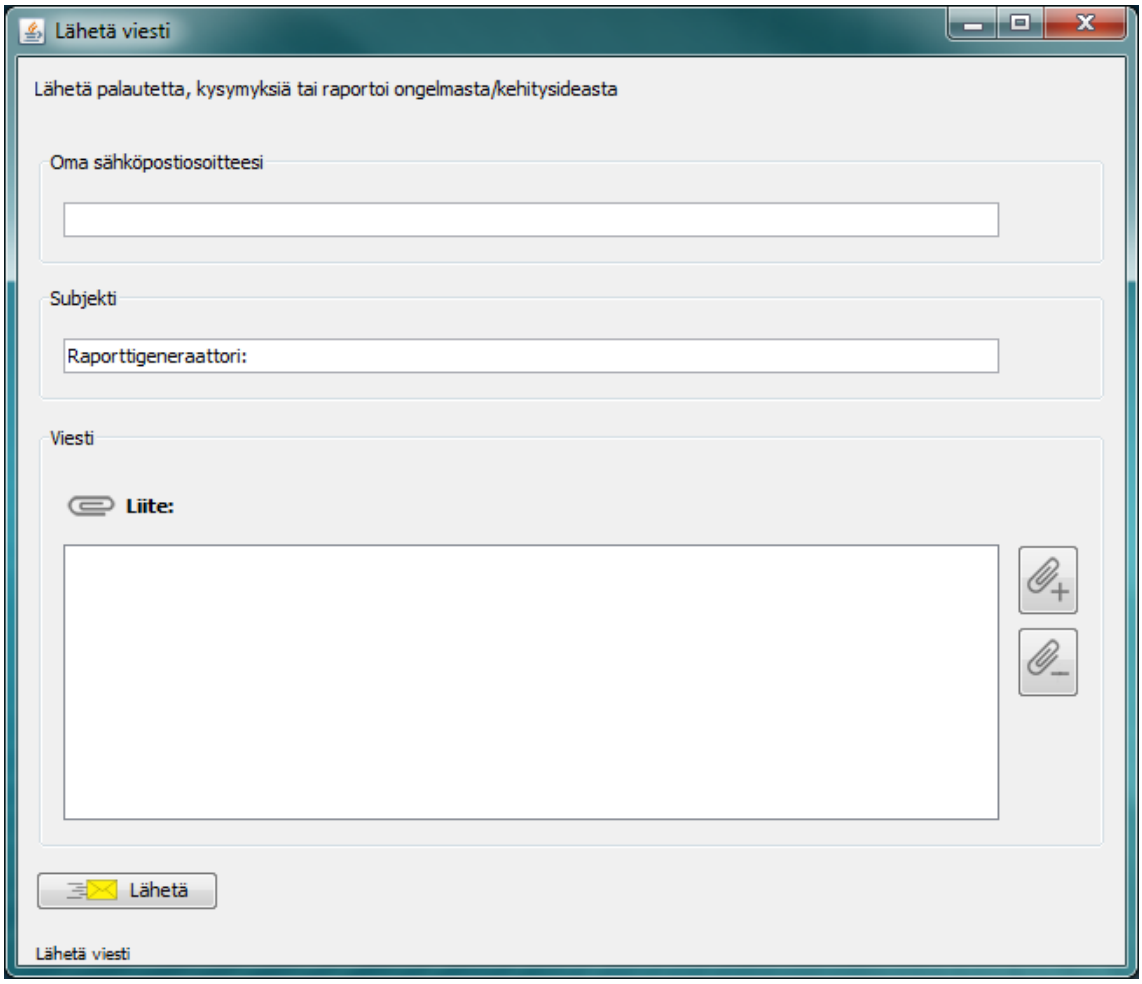
6.1.1 Help-valikko

Jos käyttäjälle tulee jokin kehitysidea tai ongelma raporttigeneraattoria käyttäessä, niin hän voi lähettää sähköpostiviestin sovelluksen kehittäjälle. Valikkorivistä Help-valikon alta löytyy valikkokohta ”Lähetä viesti” (kuva 23). Kun kohtaa napsauttaa, niin aukeaa erillinen ikkuna sähköpostiviestin lähetystä varten (kuva 24).



Kuva 23. Raporttigeneraattorin Help-valikko.

Ikkunassa syötetään oma sähköpostiosoite, viestin subjekti, viesti sekä lisätään liite, jos sellaisen haluaa viestiin liittää. Ohjelma tarkistaa, ettei sähköpostiosoite- ja subjektikenttä jää tyhjiksi. Mikäli kentät ovat tyhjiä, kun käyttäjä yrittää lähettää viestiä, niin ohjelma ilmoittaa, että sähköpostiosoite- ja subjektikenttä pitää täyttää ennen kuin viestin lähetys onnistuu.



Kuva 24. Sähköpostiviestin lähetysikkuna

Viestin lähetys on toteutettu käyttäen JavaMail API:a [5.5.4], joka tarjoaa kirjastot sähköpostiviestin muodostamista ja lähettämistä varten. Viestin muodostaminen aloitetaan määrittelemällä viestin ominaisuudet, joita ovat palvelin, käyttäjänimi ja salasana sekä istunto, jolle annetaan parametriksi edellä määritellyt ominaisuudet (kuva 25).

```
Properties properties = System.getProperties();

properties.setProperty("mail.smtp.host", host);
properties.setProperty("mail.user", "käyttäjätunnus");
properties.setProperty("mail.password", "salasana");

Session session = Session.getDefaultInstance(properties);
```

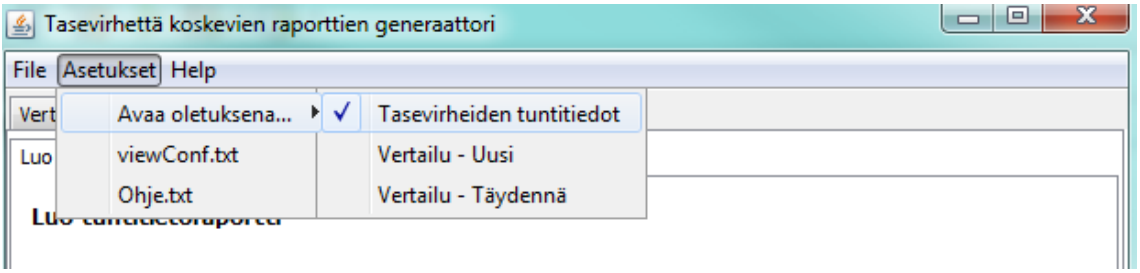
Kuva 25. Sähköpostiviestin ominaisuudet

Tämän jälkeen luodaan MimeMessage-olio eli itse viesti, johon lisätään viestin ominaisuudet sisältävä istunto sekä viestin lähettäjä, vastaanottaja, subjekti ja liite. Lopuksi

viesti lähetetään Transport.send()-metodilla, jolle annetaan parametriksi aiemmin luotu MimeMessage-olio.

6.1.2 Asetukset-valikko

Asetukset-valikosta löytyy alivalikko ”Avaa oletuksena…”, jossa päästään vaikuttamaan siihen, mikä välilehti valitaan, kun ohjelma käynnistetään (kuva 26). Näin ollen käyttäjä pystyy helpottamaan ohjelman käyttämistä. Jos käyttäjä tietää käyttävänsä ohjelman tiettyä osaa selvästi enemmän kuin muita, niin hän voi asettaa sen osan välilehden valituksi, kun ohjelma avataan.



Kuva 26. Avaa.oletuksena…-valikko

Kuvassa 26 on esitetty tilanne, jossa käyttäjä on valinnut Tasevirheiden tuntitiedot-välilehden avattavaksi oletuksena. Asetukset valikosta löytyvät myös Ohje.txt-valikkokohta, joka avaa käyttäjälle ohjelman ohje tiedoston notepad++ sovelluksessa.

6.1.3 File-valikko

File-valikko sisältää vain kohdan, jolla voi sulkea sovelluksen. Tämä valikko, tulee muuttumaan ohjelman tulevissa versioissa, mutta sen kehitys tässä työssä on jätetty minimaaliseksi.

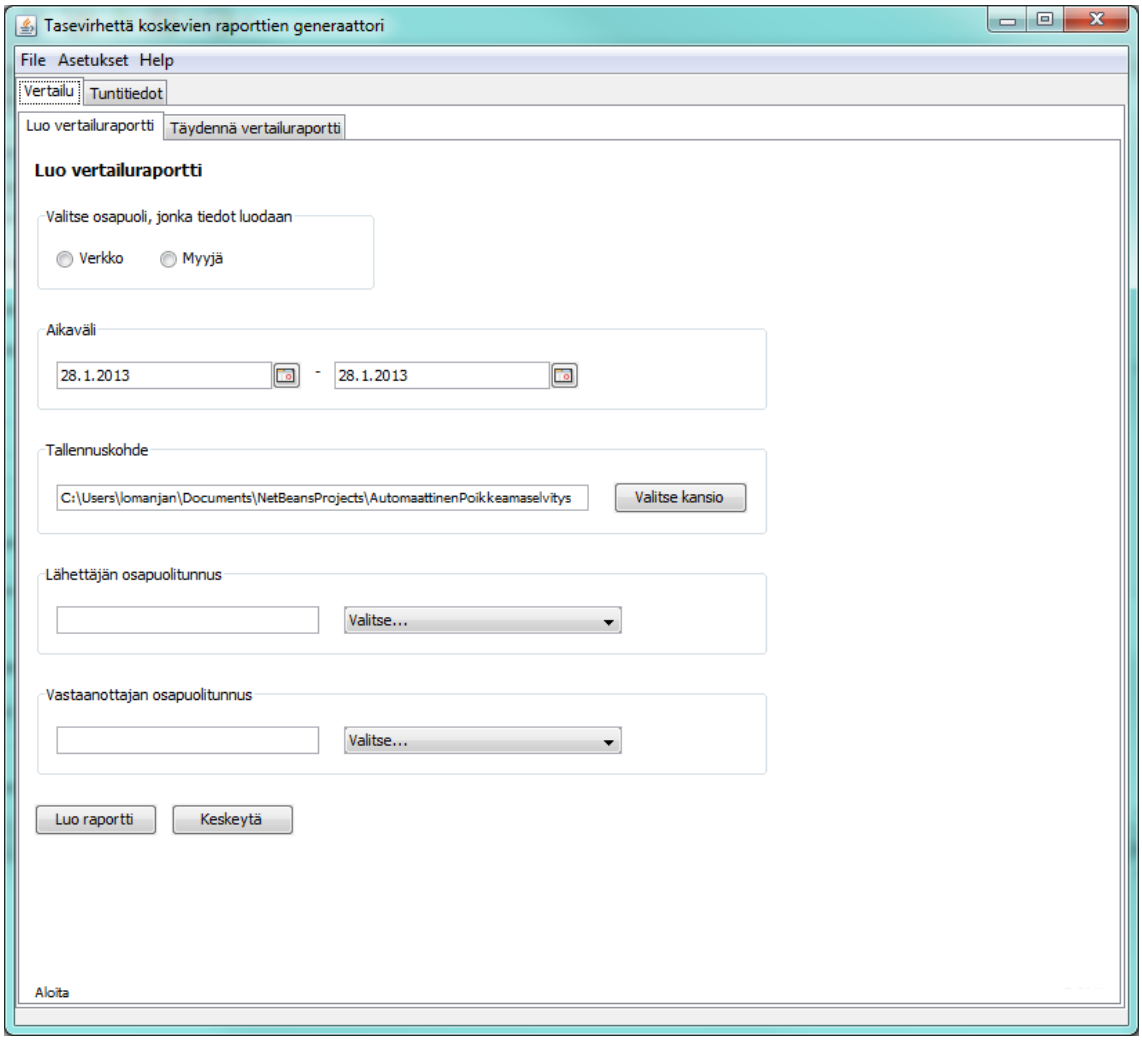
6.2 Vertailuraportin toteutus

Vertailu raportti on tehty myyjälle apuvälineeksi tasekorjauslaskun tarkistuksessa tasevirheiden löytämiseksi. Myyjä pyytää verkonhaltijalta raporttia, jos havaitsee eroja omissa ja verkonhaltijan toimittamissa tiedoissa ja epäilee tasekorjauksen virheettö-

myyttä. Raportti helpottaa myyjän ja verkonhaltijan tietojen eroavuuksien paikantamista.

Toteutusta lähdettiin rakentamaan tutustumalla Excel-tiedostojen käsittelyyn Java-ohjelmointikielellä. Ratkaisuksi valittiin Apache POI -kirjasto, joka tarjoaa kaikki tarvittavat työkalut, jotta tietokannoista kerätty tieto saadaan Energiateollisuus ry:n asettamien formaattien määräämään muotoon.

Kun Excel-tiedostojen käsittely onnistui, siirryttiin miettimään toteutuksen käyttöliittymää, jonka avulla saadaan sovelluksen käyttäjältä raportin luontiin tarvittavat tiedot. Käyttöliittymä jaettiin kahdelle välilehdelle. Toisella välilehdellä luodaan vertailuraportti ja toisella taas täydennetään vertailuraportti. Seuraavaksi esitellään käyttöliittymä (kuva 27) vertailuraportin luontia ja täydennystä varten.



Kuva 27. Luo vertailuraportti -välilehti, jolla luodaan uusi vertailuraportti.

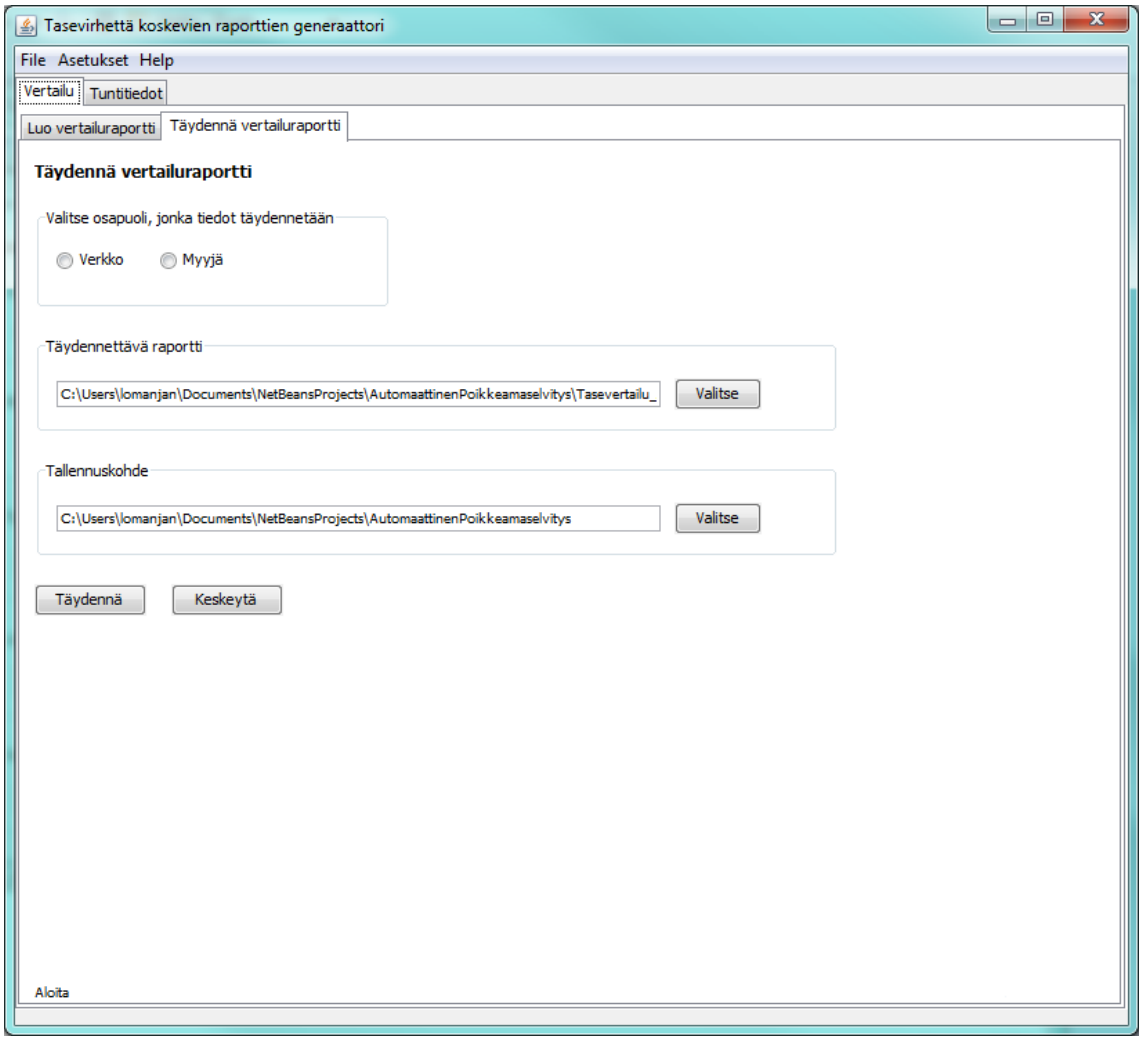
Vertailuraportin luonnissa (kuva 27) käyttäjää vaaditaan valitsemaan osapuoli, jonka tiedot luodaan, aikaväli, jonka tietoja halutaan verrata, tallennuskohde, lähettäjän osapuolitunnus sekä vastaanottajan osapuolitunnus. Jos jotakin edellä mainittua tietoa ei ole valittu käyttäjän painaessa ”Luo”-painiketta, niin ohjelma ilmoittaa, ettei kaikkia vaadittavia tietoja ole valittu. Ohjelma tarkistaa myös, voidaanko tallennuskohteeseen tallentaa. Ohjelman käyttäjällä ei välttämättä ole oikeuksia tallentaa kansioon, jonka hän on valinnut.

Kun käyttäjä painaa Luo raportti -painiketta, niin ohjelma hakee ensin PostgreSQL-tietokannasta Smac API -rajapinnalle tehtävään kyselyyn tarvittavia tietoja sekä raportissa tarvittavia käyttöpaikka- ja sopimusten voimassaolotietoja. Ohjelma käyttää PostgreSQL-tietokannan käsittelyyn, ohjelmointiympäristö NetBensista valmiiksi löytyvää, PostgreSQL JDBC -ajuria. Kun PostgreSQL-tietokannasta on saatu kaikki tarvittavat

tiedot, niin ohjelma lähettää rivinvaihdolla erotettuja JSON-objekteja sisältävän http post -pyynnön avulla kyselyn Smac API -rajapinnalle, joka muotoilee kyselyn ja lähettää sen Riakille. Riak palauttaa kyselyn tuloksen Smac API -rajapinnalle, joka muotoilee ja lähettää sen takaisin ohjelmalle. Tämän jälkeen kaikki raportin muodostamiseen tarvittavat tiedot on haettu.

Seuraavaksi ohjelma luo ilmentymän HSSFWorkbook-oliosta, joka edustaa xls-tiedostoa, johon raportti kirjoitetaan. Oliolle lisätään MYYJÄ- ja VERKKO-välilehdet. Valitun osapuolen välilehdelle asetetaan otsikkotiedot sekä otsikkotietoja vastaavat mittaustiedot, jotka on haettu tietokannoista. Tämän jälkeen välilehden soluille asetetaan tyylit ja kaavat, jotta luotava raportti vastaisi Energiateollisuus ry:n raportille asettamaa formaattia. Tyylien luomisessa käytetään HSSFCellStyle-oliota, jotka asetetaan solukohtaisesti. Lopuksi HSSFWorkbook-olio kirjoitetaan käyttäjän valitsemaan tallennuskohteeseen write()-metodilla, joka saa parametrikseen FileOutputStream-olion, jolle on asetettu tallennuskohde ja luotavan tiedoston nimi. Ohjelma avaa näytölle juuri kirjoitetun tiedoston eli valmiin vertailuraportin, joka on Energiateollisuus ry:n formaatin mukainen xls-tiedosto.

Vertailuraportin täydennyksessä (kuva 28) käyttäjää vaaditan valitsemaan kolme seikkaa: täydennetäänkö verkon vai myyjän tiedot, täydennettävä vertailuraportti sekä vertailuraportin tallennuskohde. Mikäli jotakin edellä mainituista kohdista ei ole täytetty, kun käyttäjä painaa ”Täydennä”-painiketta, niin ohjelma ilmoittaa, ettei kaikkia vaadittavia kenttiä ole täytetty eikä täydennä raporttia. Ohjelma tarkistaa myös, onko tallennuskohteeseen mahdollista tallentaa ja löytyykö valitusta vertailuraportista oikeat tiedot. Ohjelma ei täydennä raporttia ennen kuin tarvittavat tiedot on täytetty.



Kuva 28. Täydennä vertailuraportti -välilehti, jolla täydennetään olemassa oleva vertailuraportti.

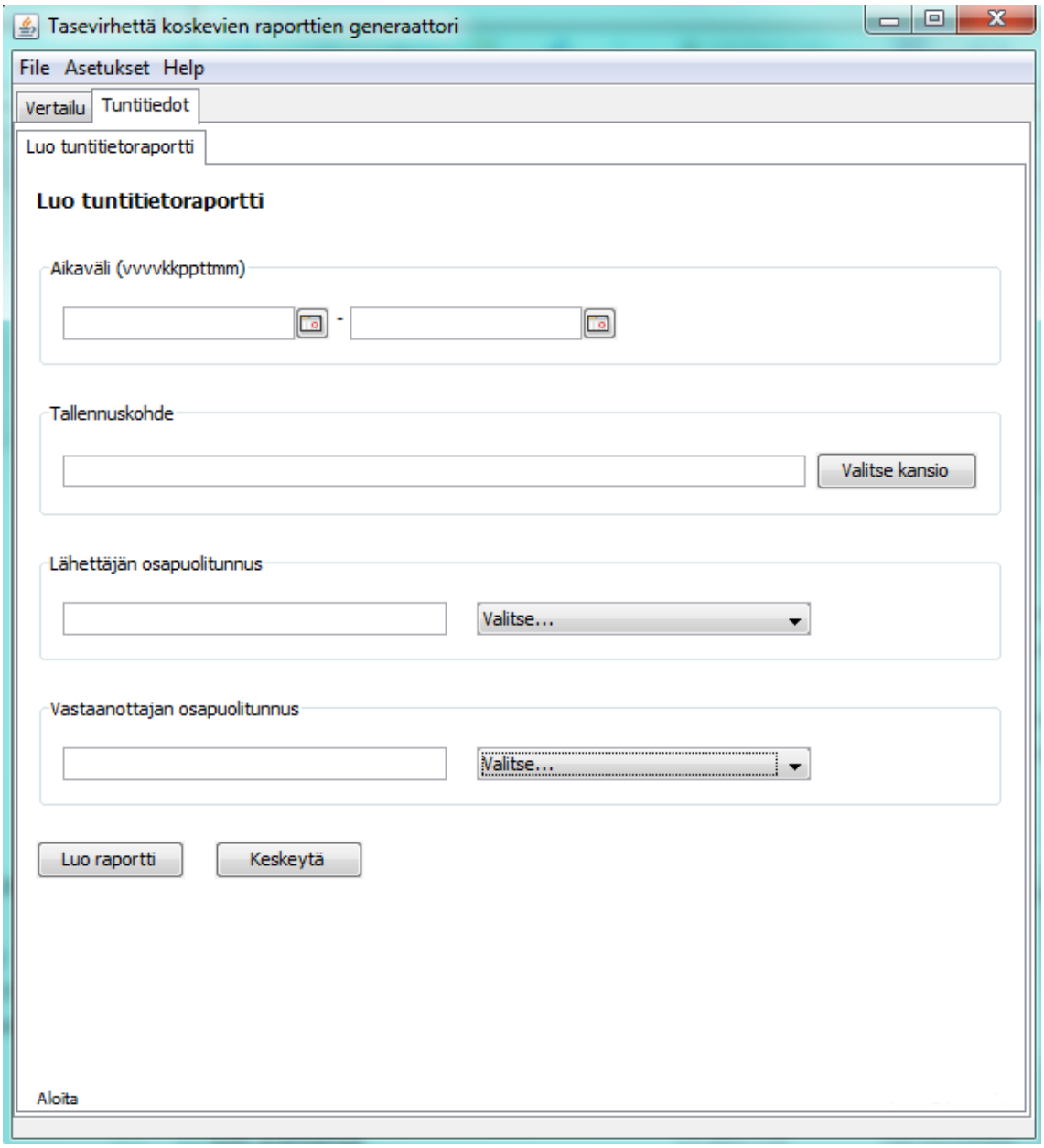
Vertailuraportin täydennys ei eroa vertailuraportin luonnista muuten kuin että siinä lähettäjän ja vastaanottajan osapuolitunnukset, aikaväli ja tiedoston nimi saadaan täydennettävästä vertailuraportista sekä Excel-tiedostoa edustavaa oliota luodessa käytetään HSSFWorkbook-olion sijasta Workbook-oliota, joka luo ilmentymän täydennettävästä vertailuraportista create-metodin avulla. Metodi saa parametrikseen InputStream-olion.

Vertailuraportin luonnissa ja täydennyksessä käytettiin useita Apache POI -kirjaston työkaluja. Edellä mainittujen HSSFWorkbook-, Workbook- ja HSSFCellStyle-olioiden lisäksi työssä käytettiin HSSFSheet- ja Sheet-olioita välilehtien käsittelyyn, HSSFRow- ja Row-olioita rivien käsittelyyn sekä HSSFCell- ja Cell-olioita solujen käsittelyyn. Näillä työkaluilla onnistuu tärkeimmät ja yleisimmät toiminnot, joita Excel-tiedostojen käsittelyssä tarvitaan.

6.3 Tasevirheiden tuntitiedot -raportin toteutus

Tasevirheiden tuntitiedot -raportin avulla verkkoyhtiö ilmoittaa myyjälle taseiden sulkeutumisen jälkeen korjatut käyttöpaikkakohtaiset tuntiaikasarjat sekä näiden aiheuttamat euromääräiset muutokset tunneittain ajanjaksolta, jota raportti käsittelee.

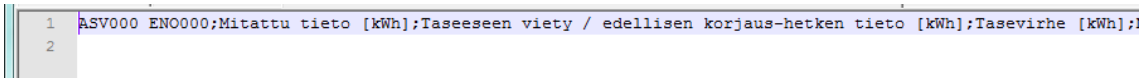
Raportin toteutusta ei saatu täysin valmiiksi, koska Smac API -rajapinnasta ei saatu raportin luontiin tarvittavia tietoja. Raportin toteutuksesta on valmiina kaikki muu paitsi tietojen haku Smac API -rajapinnan kautta Riakista sekä haettujen tietojen tallentaminen csv-tiedostoon. Seuraavaksi käydään läpi raportin luonnissa käytettävä käyttöliittymä (kuva 29).



Kuva 29. Luo tuntitietoraportti -välilehti, jonka avulla luodaan tasevirheiden tuntitietoraportti.

Käyttöliittymässä käyttäjää pyydetään valitsemaan aikaväli, jolta etsitään taseikkunan sulkeutumisen jälkeen muuttuneita tietoja. Aikaväli tulee myös osaksi tiedoston nimeä, johon tulee aikavälin lisäksi lähettäjän ja vastaanottajan osapuolitunnukset. Osapuolitunnukset valitaan kuvassakin näkyvistä alasvetovalikoista. Niitä käytetään tiedoston nimeämisen lisäksi tietojen hakemiseen sekä raportin otsikkotiedoissa. Käyttöliittymässä valitaan vielä enne raportin luontia tallennuskohde, joka määrittää sen, mihin raportti tallennetaan.

Kun käyttäjä painaa Luo raportti -painiketta, niin ohjelma tarkistaa, onko kaikki kohdat täytetty. Jos kaikkia kohtia ei ole täytetty, niin ohjelma ilmoittaa, että kaikki kentät on täytettävä, ja keskeyttää raportin luonnin. Jos kaikki kentät on täytetty, niin ohjelma siirtyisi vaiheeseen, jossa haetaan raporttiin tarvittavat tiedot. Tietojen haku on kuitenkin toteuttamatta, joten ohjelma kirjoittaa vain raportin otsikkotiedot csv-tiedostoon (kuva 30) ja tallentaa sen käyttäjän valitsemaan paikkaan.



Kuva 30. Tuntitietoraportti, jossa on vain otsikkotiedot

Csv-tiedostot ovat taulukkomuotoisia tiedostoja, joissa tiedot erotetaan puolipisteillä. Taulukossa yhdelle riville tulevat tiedot tulevat myös csv-tiedostoon yhdelle riville. Kuten kuvasta 30 voi nähdä, niin raporttiin tallennetut otsikkotiedot tulevat csv-tiedoston ensimmäiselle riville puolipisteillä eroteltuina. Csv-tiedostoja on helppo käsitellä esimerkiksi Excel-taulukkolaskentaohjelmalla.

7 Testaus

Ohjelmia ohjelmoitaessa ei pidä luottaa siihen, että ohjelmoija tuottaa virheetöntä koodia, koska jokainen tekee virheitä. Tästä syntyy tarve testata ohjelmankoodin toimivuus. Testauksella ei pyritä tekemään ohjelmasta virheetöntä, joka on usein mahdotonta, vaan sillä varmistetaan, että ohjelma toimii ainakin tietyissä tilanteissa oikein. Testausta voidaan käyttää myös apuvälineenä, kun todistetaan asiakkaalle, että ohjelma tekee sitä, mitä on sovittu.

Tässä työssä on käytetty testaamiseen Java-yksikkötestauskehystä JUnit 4 [5], joka mahdollistaa toistettavien testien kirjoittamisen. JUnit on liitettynä NetBeans-ohjelmointiympäristöön, jolloin testien kirjoittaminen ohjelmoinnin kanssa rinnakkain on helppoa. Kirjoitettavat testit ovat testattavan ominaisuuden luokan testiluokassa. Nämä kaikki testiluokat kerätään yhteen TestSuite-luokkaan, jonka avulla kaikki testit saadaan ajettua kerralla. Tässä työssä käytetty TestSuite-luokka on esitelty kuvassa 31.

```
@RunWith(Suite.class)
@Suite.SuiteClasses({Raportit.TuntitietoRaporttiTest.class,
    Raportit.VertailuRaporttiTest.class, Raportit.SahkopostiTest.class})
public class TestSuite {
    /*
```

Kuva 31. TestSuite-luokka

Kuvan 31 TestSuite-luokka testaa testiluokat TuntitietoRaporttiTest, VertailuRaportti-Test ja SahkopostiTest. Näissä testiluokissa testataan raporttien luontimetodien onnistumista, sekä palautteen lähettämismetodin onnistumista. Seuraavassa kuvassa 32 on testattu sähköpostin lähettäminen, kun lähettäjän sähköpostiosoite ja sähköpostiviestin subjekti on annettu sekä silloin, kun subjektia ei ole annettu.

```
@Test
public void testSendEmail() {
    Sahkoposti sposti = new Sahkoposti();
    boolean result;
    boolean expResult;
    if(sposti.getjTextField1().getText().compareTo("") == 0 ||
            sposti.getjTextField2().getText().compareTo("") == 0){
        sposti.getjTextField1().setText("Testi");
        sposti.getjTextField2().setText("Testi subjekti");
        sposti.getjButton1().doClick();
        result = sposti.getLahetetty();
        expResult = true;
    }
    else{
        sposti.getjButton1().doClick();
        result = sposti.getLahetetty();
        expResult = false;
    }

    assertEquals(expResult, result);
}
```

Kuva 32. Testimetodi testSendEmail()

Kuvan 32 testimetodissa lähdetään liikkeelle luomalla ilmentymä luokasta, joka sisältää testattavan ominaisuuden eli sähköpostin lähettämisen. Sitten määritellään tapaus, jossa lähettäjän sähköpostiosoite ja sähköpostiviestin subjekti on täytetty. Tapauksessa asetetaan odotettu tulos, suoritetaan metodi, poimitaan metodin tulos talteen ja suoritetaan sähköpostin lähettäminen. Tämän jälkeen määritellään tapaus, jossa lähettäjän

sähköpostiosoitetta ja sähköpostinsubjektia ei ole täytetty. Tapauksessa suoritetaan vain sähköpostiviestin lähettäminen, tuloksen talteen poimiminen ja odotetun tuloksen asettaminen. Lopuksi testimetodissa tarkistetaan olivatko saatu tulos ja odotettu tulos samoja. Jos tulokset olivat samoja, niin testimetodi läpäisee testin. Jos testimetodi läpäisee testin, niin sähköpostiviestin lähettäminen toimii, kuten toiminnallisessa määrittelyssä on määritetty. Kun ohjelma on ohjelmoitu kokonaan ja kaikki testit on kirjoitettu, niin kaikkien testien pitäisi mennä läpi. Näin varmistutaan, että ohjelma toimii niin kuin on määritetty.

JUnit 4:n lisäksi työssä on käytetty manuaalista testaamista, jossa raporttigeneraattorin tuottamien raporttien tietoja sekä sen tietokannoista keräämiä tietoja vertaillaan manuaalisesti tietokantojen tietoihin, jotta varmistutaan raporttigeneraattorin tuottavan oikeanlaista tietoa.

8 Riak-tietokannan soveltuvuus Rejlers Oy:n ja raporttigeneraattorin tietolähteeksi

Tässä osiossa perehdytään tämän työn toiseen tavoitteeseen eli arvioidaan sitä, miten Riak-tietokanta soveltuu Rejlers Oy:n ja raporttigeneraattorin tiedon lähteeksi verrattuna muihin Riakia yleisimpiin vaihtoehtoihin kuten PostgreSQL-tietokantaan.

Tällä hetkellä tiedonhallintaratkaisuja hallitsevat relaatiotietokannat. Tämän hetken kuusi suosituinta tiedonhallintajärjestelmää ovat relaatiotietokantoja. Rejlers Oy on käyttänyt aiemmin PostgreSQL-tietokantaa. Se on yksi käytetyimmistä tiedonhallintajärjestelmistä. Se on tällä hetkellä neljänneksi suosituin DB-Engines-sivuston [6] mukaan, jonne kerätään tietoa tiedonhallintajärjestelmistä. Sitä suositumpia ovat vain Oracle, MySQL sekä Microsoftin SQL Server ja Access. Vertailun vuoksi Riak löytyy vasta sijalta 22.

Jos tarkkaillaan vain tiedonhallintajärjestelmän suosion määrää, niin saattaa ihmetellä, miksi Rejlers Oy on siirtymässä mittaustietojen osalta suositusta PostgreSQL-tietokannasta Riak-tietokantaan. Tiedonhallintajärjestelmän valinnassa ei kuitenkaan lähdetä liikkeelle sen suosiosta. Sen sijaan tärkeimpinä valintakriteereinä ovat suorituskyky, kustannukset ja soveltuvuus yrityksen tarpeisiin.

NoSQL-tietokannat ovat tällä hetkellä kovassa kasvussa ja ne ovat koko ajan valtaamassa isompaa jalansijaa relaatiotietokantojen hallitsemasta tiedonhallintajärjestelmien maailmasta. NoSQL-tietokantojen kilpailuvaltteja relaatiotietokantoja vastaan ovat suorituskyky ja kustannukset. Nykyään yritykset eivät halua tinkiä suorituskyvystä, jolloin kustannukset nousevat suureen arvoon tiedonhallintajärjestelmää valittaessa. NoSQL-tietokantoja käyttää muun muassa yritykset kuten Facebook, Twitter, Amazon, LinkedIn ja Google [2].

Tietomääriltään suurissa tietokannoissa relaatiotietokantojen kustannukset nousevat ja rakenne tulee hyvin raskaaksi relaatioiden vuoksi. Näin ollen NoSQL-tietokannoille kuten Riak, tarjoutuu mahdollisuus tarjota suorituskykyä hyvään hintaan. Ei kuitenkaan pidä unohtaa, että NoSQL-tietokannat eivät ole vielä suuressa suosiossa, joten monilla yrityksillä niiden käyttöönotto voi olla suuren kynnyksen takana. Entuudestaan tuntematon järjestelmä, joka ei ole suosituimpien tiedonhallintajärjestelmien listalla voi olla suuri epävarmuustekijä. Lisäksi aina on muutosvastarintaa eli ihmisen taipumusta vastustaa asian muuttumista, johon on totuttu.

Yritys on kuitenkin valinnut Riakin, vaikka se on yritykselle entuudestaan tuntematon tiedonhallintajärjestelmä. Kuten aiemmin mainittiin, niin Reljelrs Oy on siirtymässä mittaustietojen osalta PostgreSQL:stä Riakin käyttöön. Seuraavassa taulukossa vertaillaan näiden kahden tiedonhallintajärjestelmän ominaisuuksia (taulukko 7).

Taulukko 7. Riak ja PostgreSQL -tietokantojen ominaisuuksien vertailutaulukko [6].

Nimi	Riak	PostgreSQL
Kuvaus	Hajautettu, vikasietoinen avain-arvovarasto	Oliopohjainen relaatiotietokanta
Developer	Basho Technologies	PostgreSQL Global Development Group
Ensimmäinen julkaisu	2009	1989
Lisenssi	Avoin lähdekoodi	Avoin lähdekoodi
Toteutuskieli	Erlang	C
Palvelinkäyttöjärjestelmät	Linux OS X	HP-UX Linux OS X Solaris Unix Windows
Tietokantamalli	Avain-arvo varasto	Relaatiotiedonhallintajärjestelmä
Tietorakenne	vapaa rakenne	kyllä
Ennalta määrätyt tietotyypit (float, date)	ei	kyllä
Tuetut ohjelmointikielet	C C# C++ Clojure Dart Erlang Go Groovy Haskell Java JavaScript Lisp Perl PHP Python Ruby Scala Smalltalk	.NET C C++ Java Perl Python Tcl
Viiteavaimet	Ei	Kyllä

Kuten taulukosta 7 huomaa, niin Riak on paljon rajoittuneempi palvelinkäyttöjärjestelmän valinnassa kuin PostgreSQL. Tämä ei ole kuitenkaan haitannut yritystä, koska aikaisemmin käytössä olleessa PostgreSQL-tietokannassa oli myös Linux-palvelinkäyttöjärjestelmä. Tietokantojen tietokantamalli ja tietorakenne eroavat hyvin paljon toisistaan, sillä toinen on SQL-tietokanta eli relaatiotietokanta ja toinen NoSQL-tietokanta. PostgreSQL:ssä tiedot ovat liitettynä relaatioiden avulla toisiinsa, kun taas Riakissa tiedot on tallennettuna avain-arvo pareina, joilla on vain linkkejä niihin liittyviin

asioihin. Kun tarkkaillaan tietokantojen tuettuja ohjelmointikieliä, niin Riak erottuu edukseen hyvin kattavalla kielivalikoimalla. Molemmat tietokannat ovat avoimen lähdekoodin lisenssin alla. Tämä ei välttämättä tarkoita sitä, että molemmat ovat ilmaisia käyttää, mutta tässä tapauksessa yrityksellä on ollut molemmissa ratkaisuissa ilmaisversiot käytössä, jolloin itse tiedonhallintajärjestelmä ei aiheuta kustannuksia.

Kuvan vertailutaulukko ei kuitenkaan osoita ehkä niitä tärkeimpiä syitä, miksi pitäisi valita juuri Riak tai PostgreSQL. Taulukosta puuttuu neljä hyvin tärkeää asiaa, jotka ovat suorituskyky, vikasietoisuus, kustannukset ja soveltuvuus. Nämä ehkä tärkeimmät valintakriteerit on selvitetty vertaamalla yrityksen tietoja kummankin tietokannan aiheuttamista kustannuksista, vikasietoisuudesta, suorituskyvystä sekä soveltuvuudesta.

Kuten aiemmin mainittiin, niin yrityksellä on käytössä Riakin sekä PostgreSQL:n ilmaisversiot, joten kummastakaan ei aiheudu yritykselle suoria kustannuksia. Laitteistot aiheuttavat toki kustannuksia, mutta tässä tilanteessa niistä aiheutuvat kustannukset ovat yhtä suuret. Molemmat tiedonhallintajärjestelmät soveltuvat rakenteeltaan yhtä hyvin yrityksen käyttötarkoituksiin. Tästä seuraa tilanne, jossa kustannukset ja soveltuvuus eivät vaikuta tietokannan valintaan vaan siihen vaikuttaa tietokannan suorituskyky nykyisillä kustannuksilla sekä vikasietoisuus, joka on Riakin yksi vahvuuksista.

Tietokannan suorituskyky valaisee hyvin syytä, miksi yritys päätyi valitsemaan Riakin PostgreSQL:n korvaajaksi. Käyttökokemusten perusteella Riakin kirjoitusoperaatiot ovat 1000 kertaa nopeampia kuin PostgreSQL:n kirjoitusoperaatiot. Tällä hetkellä PostgreSQL:n tarvitsema aika kirjoitusoperaatioihin alkaa olla äärirajoilla. Ollaan tulossa tilanteeseen, jossa lukemia ei saada tallennettua ja lähetettyä ajallaan. Lisäksi lukemien määrä tietokannassa kasvaa kovaa vauhtia tuntimittauksiin siirtymisen ja uusien asiakkaiden vuoksi, jolloin PostgreSQL:n tukalaan tilanteeseen ei näy olevan tulossa helpotusta.

Vaikka Riak soveltuu yrityksen tietolähteeksi, niin se ei tarkoita sitä, että Riak olisi parempi ratkaisu raporttigeneraattorin näkökulmasta. Raporttigeneraattorille se tarkoittaa rajapinnan rakentamista Riakin ja raporttigeneraattorin välille sekä kerättävän tiedon jakautumista kahteen tietokantaan. Lisäksi tietojen saaminen, Smac API -rajapinnan takana olevasta Riakista, vaatii aputietojen hakemista PostgreSQL-tietokannasta, joka mutkistaa entisestään raporttigeneraattorin rakennetta. Kuitenkin jos ohjelman halutaan toimivan nopeasti ilman pidempiä odotteluita, on Riak parempi ratkaisu. Haku Riakista

on muutenkin paljon nopeampaa kuin PostgreSQL:stä, mutta Riakkiin on lisäksi tallennettu tieto taseikkunan sulkeutumisen jälkeen muuttuneista arvoista, mikä tiputtaa läpikäytävän tiedon määrän murto-osaan siitä mitä se olisi ilman tätä tietoa. PostgreSQL-tietokannassa tämänkaltaista tietoa ei ole, jolloin hakua ei pysty rajaaman lähellekään yhtä tehokkaasti kuin Riakissa.

Näiden vertailujen jälkeen voi sanoa sen, että Riak soveltuu sekä raporttigeneraattorin, että Rejlers Oy:n tietolähteeksi todella hyvin. Raporttigeneraattorin toteutukseen se aiheuttaa hieman lisätyötä, mutta ohjelman toimivuutta se parantaa huomattavasti. Rejlers Oy:n kannalta tietolähteen vaihtaminen NoSQL-tietokantaan on lähes välttämätöntä ilman, että kustannukset nousisivat, koska talletettavan tiedon määrä on suuressa kasvussa ja PostgreSQL on jo äärirajoillaan suorituskyvyn kanssa.

9 Yhteenveto

Tässä insinöörityössä toteutettiin raporttigeneraattori, joka tuottaa Energiateollisuus ry:n asettamien formaattien mukaisia tasevirheraportteja, sekä arvioitiin, miten Riak-tietokanta soveltuu Rejlers Oy:n ja raporttigeneraattorin tietolähteeksi.

Työ aloitettiin käymällä läpi työn tavoitteet ja työssä käytetyt teknologiat ja menetelmät. Tässä yhteydessä esiteltiin PostgreSQL, Riak-tietokanta ja sen rakenne sekä käytetyt suunnittelumallit. Sen jälkeen työssä siirryttiin määrittelyvaiheeseen, joka jaettiin toiminnalliseen ja tekniseen määrittelyyn. Toiminnallisessa määrittelyssä kerrottiin työn vaatimukset sekä ohjelman toiminnallisuus. Teknisessä määrittelyssä käytiin läpi ohjelman arkkitehtuuria, rajapintoja, ohjelmointikieli, ohjelmointiympäristö sekä tärkeimmät ohjelman teossa tarvitut kirjastot.

Määrittelyjen jälkeen kuvattiin raporttigeneraattorin varsinainen toteutus. Toteutus jakautui kolmeen osaan. Yhdessä osassa kuvattiin raporttigeneraattorin hallinnan toteutusta ja kahdessa muussa itse raporttien toteutusta. Raporttien toteutus jakautui tuntitietoraporttien toteutukseen ja vertailuraporttien toteutukseen. Toteutuksien jälkeen kerrottiin vielä yleisesti testauksesta ja siitä, millä työkaluilla ja miten raporttigeneraattoria testataan.

Työn tavoitteita tarkennettiin työn edetessä, kun huomattiin, että ulkopuolisista tekijöistä johtuen tasevirheiden tuntitieto -raporttiin tarvittavien tietojen hakemista ei ehdi toteuttaa. Lisäksi vertailuraportin toteutuksessa päädyttiin viime hetkellä muuttamaan Smac API -rajapinnan toteutusta, joten vertailuraportteja ei vielä voida ottaa tuotantokäyttöön. Tämä Smac API -rajapintaan tehty muutos on tämän työn tuloksena syntynyt kehitysidea, joka parantaa huomattavasti raporttigeneraattorin toimintaa. Vertailuraporttien tuottaminen onnistuisi ilman muutosta, mutta se olisi paljon hitaampaa ja hankalampaa. Muutoksessa osa tietojen muokkaamisesta siirrettään raporttigeneraattorilta Smac API -rajapinnalle. Työn tuloksena syntynyt raporttigeneraattori on valmis runko, joka saadaan pienin muutoksin tuotantokäyttöön.

Työn tuloksena saatiin myös arvokasta tietoa siitä, miten hyvin Riak soveltuu Rejlers Oy:n ja raporttigeneraattorin tietolähteeksi. Työ auttoi yritystä uskomaan Riakin toimivuuteen ja kannattavuuteen sekä osoitti, että Riak soveltuu raporttigeneraattorin tietolähteeksi, vaikka Riakin täyttä potentiaalia ei ole vielä saatu hyödynnettyä.

Työ onnistui hyvin. Se täytti lähes kaikki sille asetetut vaatimukset ja tavoitteet, vaikka aikataulu ja rajapintaongelmat haittasivat paljon työn valmistumista. Välillä vaikutti siltä, ettei työhön saada ollenkaan kommunikointia Smac API -rajapinnan ja sitä myöten myös Riakin kanssa. Nämä aikataulu- ja rajapintaongelmat olivat ulkopuolisista tekijöistä kiinni eikä niihin pystytty vaikuttamaan. Työn tuloksena syntynyt raporttigeneraattori ei ole valmis tuotantokäyttöön, mutta sitä tullaan jatkokehittämään, jotta se täyttäisi kaikki tässä työssä sille asetetut vaatimukset. Työ synnytti myös uusia työn aiheeseen liittyviä ohjelma- ja kehitysideoita, joista voi olla suuri apu yrityksen tuotannon laadun parantamisessa. Lisäksi työ paljasti virheitä ja epätarkkuuksia Energiateollisuus ry:n asettamista formaateista ja malliesimerkeistä.

Kaiken kaikkiaan työ opetti paljon siitä, kuinka työelämässä syntyneestä tarpeesta saadaan tehtyä valmis ohjelma. Parhaiten mieleen jäänyt opetus on se, kuinka paljon työelämässä tarvitaan joustavuutta. Kun ollaan tekemisissä useiden tahojen kanssa ja jokaisella taholla on omat aikataulunsa eikä kaikki suju aina niin kuin on suunniteltu, niin joustavuutta tarvitaan. Työ auttoi ymmärtämään sitä, miksi aikatauluihin panostamisen tärkeyttä painotetaan opinnoissa ja miksi ohjelmistoprojektit usein kestävät pidempään, kuin oli alun perin suunniteltu.

Lisäksi työn aikana tuli opittua paljon uutta Riak- ja PostgreSQL-tietokannoista sekä JSON-tiedonsiirtomuodosta ja Apache http Client -kirjastosta. Nämä asiat olivat entuudestaan tuntemattomia, mutta työn jälkeen niiden toiminta ja rakenne ovat tulleet tutuksi. Tämä työ kehitti paljon ohjelmointitaitoja, vei eteenpäin ohjelmoijana sekä antoi kuvaa omasta osaamisesta ja siitä, mitä on tarttunut mukaan opintojen varrella.

Lähteet

1. Rejlers Oy. Verkkodokumentti. <http://www.rejlers.fi>. Luettu 7.2.2013.

2. Riak home. Verkkodokumentti. <http://basho.com>. Luettu 2.1.2013.

3. Riak documents. Verkkodokumentti. <docs.basho.com>. Luettu 2.1.2013.

4. Silander, Simo. Ollikainen, Vesa. Peltomäki, Juha. 2010. Java. WSOYpro Oy.

5. JUnit. Verkkodokumentti. <http://en.wikipedia.org/wiki/JUnit>. Luettu 9.1.2013.

6. DB-Engines. Verkkodokumentti. <http://db-engines.com/en/ranking>. Luettu 11.3.2013.

7. Taseisiin jääneiden virheiden käsittely taseiden sulkeutumisen jälkeen. Verkkodokumentti. Energiateollisuus. <http://energia.fi/sites/default/files/images/tasevirheiden_kasittely_raportti_2012_liitteineen.pdf>. Luettu 2.1.2013.

8. Java. Verkkodokumentti. <http://www.java.com/en/about/>. Luettu 2.1.2013.

9. Netbeans Verkkodokumentti. <http://netbeans.org/about/index.html>. Luettu 2.1.2013.

10. NoSQL. Verkkodokumentti. <http://en.wikipedia.org/wiki/NoSQL>. Luettu 7.2.2013.

11. JavaMail API. Verkkodokumentti. < http://www.oracle.com/technetwork/java/javamail/index.html>. Luettu 8.2.2013.

12. PostgreSQL JDBC Driver. Verkkodokumentti. <http://jdbc.postgresql.org/ >. Luettu 8.2.23013.

13. ApachePOI. Verkkodokumentti. <http://poi.apache.org/>. Luettu 2.1.2013.

14. Smac API. Verkkodokumentti. < https://wiki.resqueue.com/wiki/pages/N5j7z7L4/Smac_API.html >. Luettu 8.2.2013.

15. ApachePOI:n lataus. Verkkodokumentti. <http://poi.apache.org/download.html>. Luettu 3.1.2013.

Tasevirhettä koskevien raporttien formaattien määrittely

Energiateollisuus

SÄHKÖKAUPPA
SÄHKÖVERKKO
Markus Pilspanen
4.1.2012

OHJE 1(3)

Tasevirhettä koskevien raporttien formaatin määrittely

Tasevirheiden käsittely työryhmä on yhdessä teknisen ryhmän kanssa määritellyt sisältöformaatin kahdelle määrämuotoiselle sähköiselle dokumentille, joiden tarkoitus on helpottaa taseisiin jääneiden virheiden automaattista käsittelyä taseiden sulkeutumisen jälkeen.

Tämän ohjeen tarkoitus on määritellä näiden kahden sähköisen dokumentin formaatti yksiselitteisesti siten, että jokainen toimija luo dokumentin samalla tavalla ja käsittelee dokumentissa olevat tiedot samalla tavalla.

Raportti 1: Tasevirheiden tuntitiedot

Tämän raportin avulla verkkoyhtiö ilmoittaa myyjälle taseiden sulkeutumisen jälkeen korjatut käyttöpaikkakohtaiset tuntiaikasarjat sekä näiden aiheuttamat euromääräiset muutokset tunneittain ajanjaksolta, jota raportti käsittelee.

Raportin tiedostomuoto on .CSV, joka mahdollistaa tietojen vaivattoman koneellisen luennan tietojärjestelmään sekä sisällön helpon käsittelyn yleisimmillä taulukkolaskentaohjelmilla, kuten Excelillä. Raportin käsittelyn helpottamiseksi tulee tiedoston nimen sisältää tietyt otsikkotiedot alla olevan määrittelyn mukaisesti, jossa tietojen erotinmerkkinä toimii alaviiva:

`Tasevirhetuntitiedot_JVH000_MYYJ_201204302200Z_201205012100Z_1.csv`

- Tiedoston nimi alkaa aina "Tasevirhetuntitiedot"
- Lähettäjän osapuolitunnus (esim. "JVH000")
- Vastaanottajan osapuolitunnus (esim." MYYJ")
- Aikaväli, jota raportti käsittelee. Aikavälin tulee täsmätä raportin sisällön aikaleimoihin.
    - alkuaikaleima (esim. "201204302200Z")
    - loppuaikaleima (esim. "201205012100Z")
- Sekvenssinumero (esim. "1"). Sekvenssi on juokseva luku, joka tekee tiedostosta yksilöllisesti tunnistettavan samalta aikaväliltä samalle myyjälle tietoja uudelleen lähetettäessä, jolloin nimi muuten olisi sama. Lähettäjä voi itse määritellä sekvenssin haluamansa nimeämiskäytännön mukaan, kunhan se on luku ja myöhemmin lähetetyssä samaa myyjää ja aikaväliä koskevassa tiedostossa sekvenssi on aikaisemmin lähetettyä suurempi. Sekvenssin ei siis aina tarvitse olla edellinen + 1.

Raportti on taulukkomuotoinen ja sen ensimmäinen rivi on (ensimmäistä solua lukuun ottamatta) varattu sarakkeita koskeville määrämuotoisille otsikkotiedoille, jotka on kuvattu alempana. Toisella rivillä esitetään käyttöpaikkanumero, jota sarakkeessa oleva aikasarja koskee. Yhtä käyttöpaikkaa kohden on kolme eri tietoa vierekkäisillä sarakkeilla, joten sama käyttöpaikkanumero toistuu aina kolme kertaa. Käyttöpaikkanumerot ovat suuruusjärjestyksessä pienimmästä suurimpaan. Raportin kolmannesta rivistä alkaen jokainen rivi kohdistuu tiettyyn aikaleimaan, jotka on esitetty ensimmäisessä sarakkeessa rivistä kolme alkaen. Sarakkeen aikaleima kertoo tunnin alkamisajan. Aikaleimat ovat UTC-0-aikaa ja ne on esitetty muodossa: "2012-04-30T22:00:00Z" (Huom! ET:n muussa ohjeistuksessa on vastaavissa tapauksissa pääsääntöisesti käytetty virallista aikaa). Raportti kuitenkin koostuu Suomen virallisen ajan mukaisista kokonaisista vuorokausista. Alku- ja loppuaikaleimat eivät siis ajoitu UTC-0-ajan mukaiseen vuorokauden vaihteeseen, vaan Suomen virallisen ajan mukaiseen vuorokauden vaihteeseen. Vuorokausi alkaa siis joko 22:00 tai 21:00 riippuen onko talvi- vai kesäaika.

Energiateollisuus ry
Fredrikinkatu 51–53 B, 00100 Helsinki
PL 100, 00101 Helsinki
Puhelin: (09) 530 520, faksi: (09) 5305 2900
www.energia.fi

Sarakkeiden määrämuotoiset otsikot ensimmäisellä rivillä:

- Sarake 1: "JVH000_MYYJ". Tämä solu on ensimmäisen rivin ainoa muuttuva tieto. Solu sisältää verkonhaltijan ja myyjän osapuolitunnukset otsikon mukaisessa muodossa:
- Sarake 2: "Mitattu tieto [kWh]". Tämä sarake kertoo kyseisen käyttöpaikan lopulliset mitatut tiedot niiltä tunneilta, kun tasevirhettä kyseisen käyttöpaikan osalta korjataan. Yksikkö on kWh ja tarkkuus kaksi desimaalia. Desimaalierottimena toimii pilkku. Mitatut arvot syötetään tiedostoon vain siltä osin kuin niissä on muutoksia aikaisemmin taseeseen vietyyn arvoon, joten aikasarjan ei tarvitse kattaa koko tiedoston tarkasteluaikaa. Arvojen oletetaan olevan ns. vahvalla statukselle (Korjattu-OK), koska raportilla korjataan jo aiemmin vahvalla statuksella lähetettyjä tietoja.
- Sarake 3: "Taseeseen viety / edellisen korjaus-hetken tieto [kWh]". Sarake kertoo kyseisen käyttöpaikan taseisiin viedyt energiat tai edellisen korjausaineiston tiedot niiden tuntien osalta, kun tasevirhettä kyseisen käyttöpaikan osalta korjataan. Yksikkö, tarkkuus ja desimaalierotin ovat samat kuin mitatulla tiedolla. Myös aikasarjan pituus tulee olla sama kuin mitatulla tiedolla.
- Sarake 4: "Tasevirhe [kWh]". Sarake kertoo käyttöpaikkakohtaiset tunneittaiset tasevirheet, jotka lasketaan vähentämällä mitatusta tiedosta taseisiin viety tai edellisen korjaushetken tieto. Mikäli uusi mitattu tieto on suurempi tai yhtä suuri kuin taseeseen viety tieto, on virheen etumerkki positiivinen, muussa tapauksessa etumerkki on negatiivinen. Yksikkö, tarkkuus ja desimaalierotin ovat samat kuin mitatulla tiedolla. Myös aikasarjan pituus tulee olla sama kuin mitatulla tiedolla.
- Sarakkeesta 5 alkaen toistuu sarakkeiden 2 – 4 otsikot sekä sisällöstä annetut määritykset käyttöpaikkakohtaisissa kolmen tiedon sarjoissa. Eli sarakkeissa 5, 6 ja 7 on järjestyksessä seuraavan käyttöpaikan vastaavat tiedot kuin sarakkeissa 2, 3 ja 4 jne.
- Kolmanneksi viimeinen sarake: "SPOT [EUR/MWh]". Tämä sarake sisältää Nord Poolin ilmoittaman Suomen alueen Spot-hintojen tuntiaikasarjan raportin tarkastelemalta aikaväliltä. Spot-aikasarjan tulee ehdottomasti vastata ensimmäisen sarakkeen aikaleimoja, sillä tasevirheen euromääräinen arvo lasketaan tämän tiedon perusteella rivikohtaisesti. Spot-hintojen yksikkö on €/MWh ja tarkkuus yksi sentti, eli kaksi desimaalia.
- Toiseksi viimeinen sarake "Korjaus [EUR]". Tämä sarake kertoo tasevirheistä johtuneen korjauksen euromääräisen summan jokaista aikaleimaa kohden. Tämä lasketaan kyseisen aikaleiman spot-hinnan ja eri käyttöpaikkojen tasevirheiden tulojen summana. Yksikkö on euro ja se ilmoitetaan tarkkana lukuna enintään seitsemän desimaalin tarkkuudella.
- Viimeinen sarake: "Korjaussumma yhteensä [EUR]": Raportin viimeinen sarake kertoo yhteenlasketun euromääräisen korjaussumman koko raportin tarkasteluajalta. Tämä yksittäinen lukema kirjataan sarakkeen kolmannelle riville. Korjaussumman yksikkö on euro ja se ilmoitetaan kahden desimaalin eli yhden sentin tarkkuudella. Plusmerkkinen summa kertoo, että myyjää veloitetaan korjauslaskulla, koska taseisiin on viety liian vähän energiaa ja miinusmerkkinen kertoo, että kyseistä myyjää hyvitetään taseisiin viedyn liian suuren energian johdosta.

Tiedoston manuaalisessa käsittelyssä taulukkolaskentaohjelmilla voi ilmetä rajoituksia. Esimerkiksi MS Excel 2010 mahdollistaa enintään 16 384 saraketta käsittävän taulukon käsittelyn, mikä vastaa 5460 käyttöpaikan tiedot sisältävää raporttia. Yhtä aikaväliä koskevat tiedot tulee kuitenkin aina lähettää kokonaisuudessaan yhdessä osassa, eli tietoja ei saa jakaa useaan samaa aikaväliä koskevaan raporttiin. Vastaanottajan tulee varautua siihen, että tiedosto ei välttämättä avaudu oikein jos taulukkolaskentaohjelmassa on rajoitus sarakkeiden määrän suhteen. CSV-formaatissa rajoitusta ei ole, joten suoraan raporttia koneellisesti CSV-tiedostosta lukemalla tiedot saadaan järjestelmään oikein.

Esimerkki raportista löytyy ET:n sanomaliikennesivujen kohdasta ohjeet ja suositukset, Tasevirhetuntitiedot_JVH000_MYYJ_YYYYMMDDHHMMZ_YYYYMMDDHHMMZ_1.csv

Raportti 2: Vertailu tasevirheiden löytämiseksi

Tämä verkonhaltijan ja myyjän tiedoilla täydennettävä vertailuraportti on tarkoitettu myyjälle apuvälineeksi tasekorjauslaskun tarkistuksessa. Se tulee pyytää verkonhaltijalta aina erikseen, mikäli myyjä havaitsee eroja omissa ja verkonhaltijan ilmoittamissa

tiedoissa ja epäilee tasekorjauslaskun oikeellisuutta. Raportti tarjoaa myyjälle lisäselvitystä asiasta, jotta erojen syyt voidaan paremmin paikantaa.

Kaikki raportissa esitettävät arvot noudattavat Suomen virallisen ajan mukaisia vuorokausia.

Raportin tiedostomuoto on Excel. Mallipohja löytyy ET:n sanomaliikennesivujen kohdasta ohjeet ja suositukset, Tasevertailu_JVH000_MYYJ.xls

Vertailuraportin käyttö

Erojen ja epäselvyyksien syiden löytämistä voi helpottaa tämän esimerkin mukaisella laskennalla. Taulukon ensimmäisen välilehden (=VERKKO) tuottaa verkonhaltija ja toisen myyjä itse (=MYYJÄ).

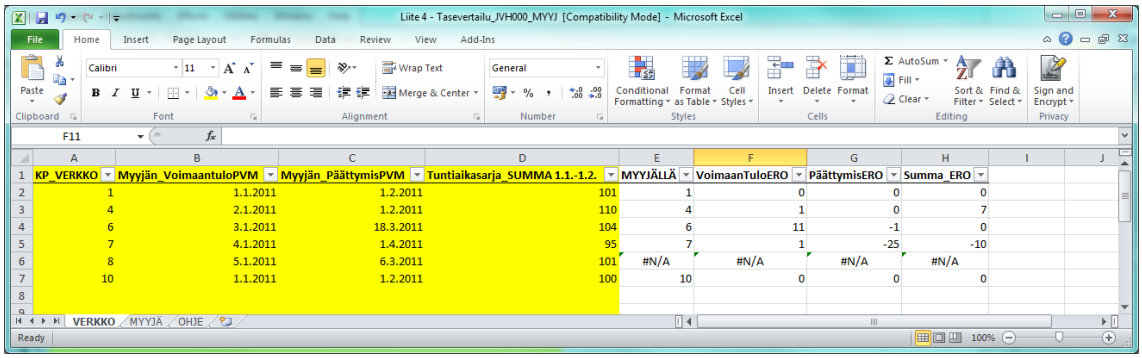
Taulukossa on keltaisella pohjalla merkitty tiedot, jotka täyttäjä hakee järjestelmistä vertailutaulukkoon. Valkoisella pohjalla on esitetty vertailun tulos.

Verkon järjestelmän tulee tuottaa ensimmäiselle välilehdelle sarakkeeseen A kaikki kyseisen myyjän hänen vastuualueellaan tuntimittauksen piirissä olevat käyttöpaikat tunnuksineen kasvavassa järjestyksessä. Sarakkeeseen B tulee ko. käyttöpaikan myyjätiedon voimaantulopäivämäärä ja sarakkeeseen C myyjätiedon päättymispäivämäärä. Voimassa olevilla sopimuksilla tämä kenttä jää tyhjäksi. Saman myyjän peräkkäiset sopimukset voidaan ketjuttaa samalle riville. Jos sopimuksien välissä on epäjatkuvuuskohta, jaetaan sopimukset eri riveille.

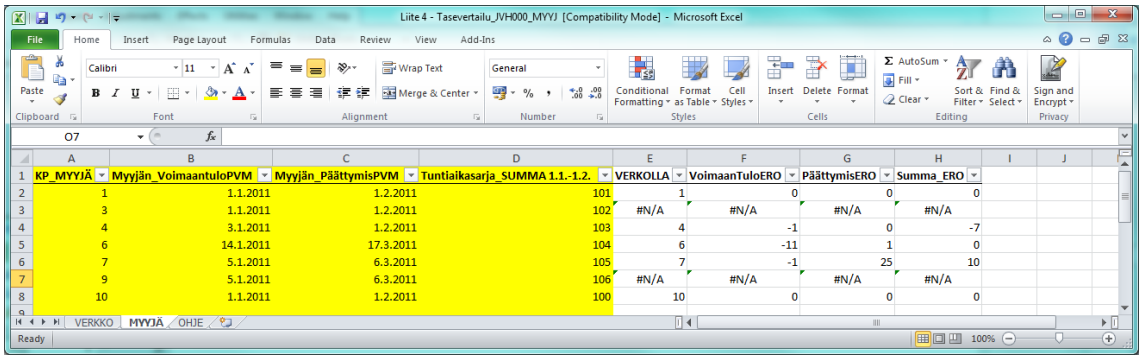
Sarakkeeseen D lasketaan vertailtavan ajanjakson, esimerkiksi vuosi tai tietty kuukausi (tarkistusotannan ollessa kyseessä), käyttöpaikkakohtainen ko. ajanjakson korjatusta tuntiaikasarjasta (ei välttämättä taseeseen viety) yhteenlaskettu sähköenergian käyttö.

Vastaavasti myyjän järjestelmä tuottaa oman näkemyksensä mukaisen vertailuaineiston välilehdelle "MYYJÄ". Tämä jälkeen näiden kahden välilehden toisiaan vastaavia soluja verrataan keskenään ja vertailun tuloksena saadaan järjestelmissä olevat eroavaisuudet.

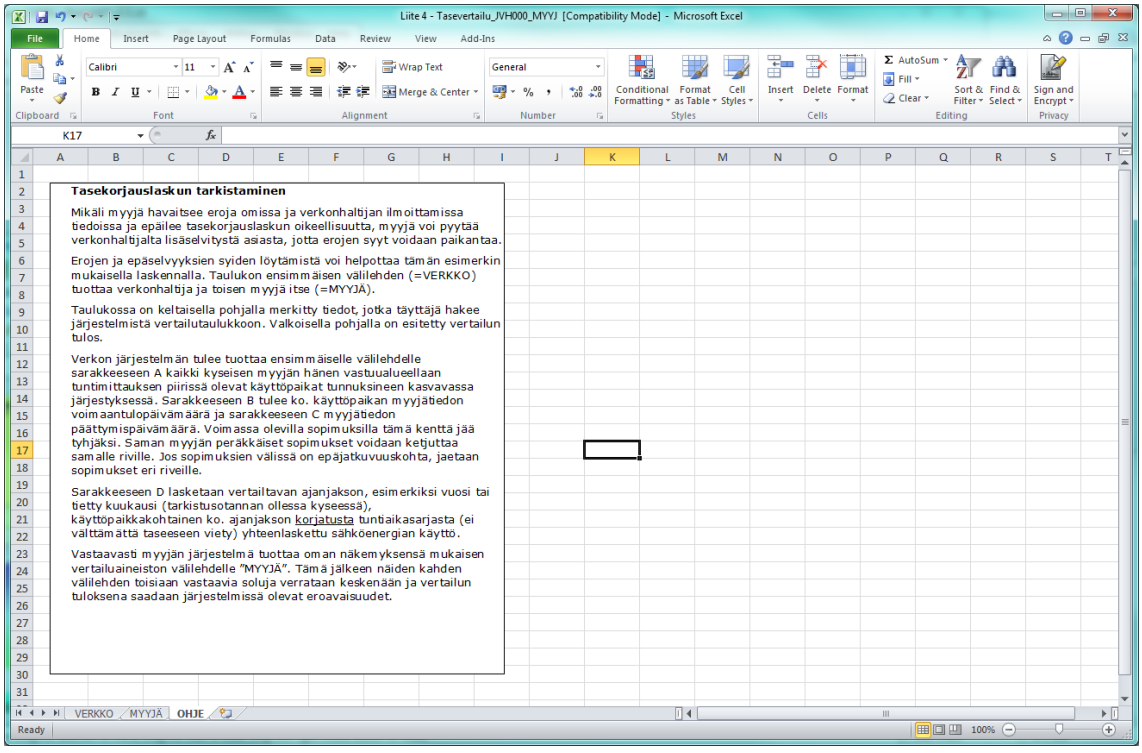
Tasevertailu-raportin malliesimerkki



KP_VERKKO	Myyjän_VoimaantuloPVM	Myyjän_PäättymisPVM	Tuntiaikasarja_SUMMA 1.1.-1.2.	MYYJÄLLÄ	VoimaanTuloERO	PäättymisERO	Summa_ERO
1	1.1.2011	1.2.2011	101	1	0	0	0
4	2.1.2011	1.2.2011	110	4	1	0	7
6	3.1.2011	18.3.2011	104	6	11	-1	0
7	4.1.2011	1.4.2011	95	7	1	-25	-10
8	5.1.2011	6.3.2011	101	#N/A	#N/A	#N/A	#N/A
10	1.1.2011	1.2.2011	100	10	0	0	0



KP_MYYJÄ	Myyjän_VoimaantuloPVM	Myyjän_PäättymisPVM	Tuntiaikasarja_SUMMA 1.1.-1.2.	VERKOLLA	VoimaanTuloERO	PäättymisERO	Summa_ERO
1	1.1.2011	1.2.2011	101	1	0	0	0
3	1.1.2011	1.2.2011	102	#N/A	#N/A	#N/A	#N/A
4	3.1.2011	1.2.2011	103	4	-1	0	-7
6	14.1.2011	17.3.2011	104	6	-11	1	0
7	5.1.2011	6.3.2011	105	7	-1	25	10
9	5.1.2011	6.3.2011	106	#N/A	#N/A	#N/A	#N/A
10	1.1.2011	1.2.2011	100	10	0	0	0



Tasekorjauslaskun tarkistaminen

Mikäli myyjä havaitsee eroja omissa ja verkonhaltijan ilmoittamissa tiedoissa ja epäilee tasekorjauslaskun oikeellisuutta, myyjä voi pyytää verkonhaltijalta lisäselvitystä asiasta, jotta erojen syyt voidaan paikantaa.

Erojen ja epäselvyyksien syiden löytämistä voi helpottaa tämän esimerkin mukaisella laskennalla. Taulukon ensimmäisen välilehden (=VERKKO) tuottaa verkonhaltija ja toisen myyjä itse (=MYYJÄ).

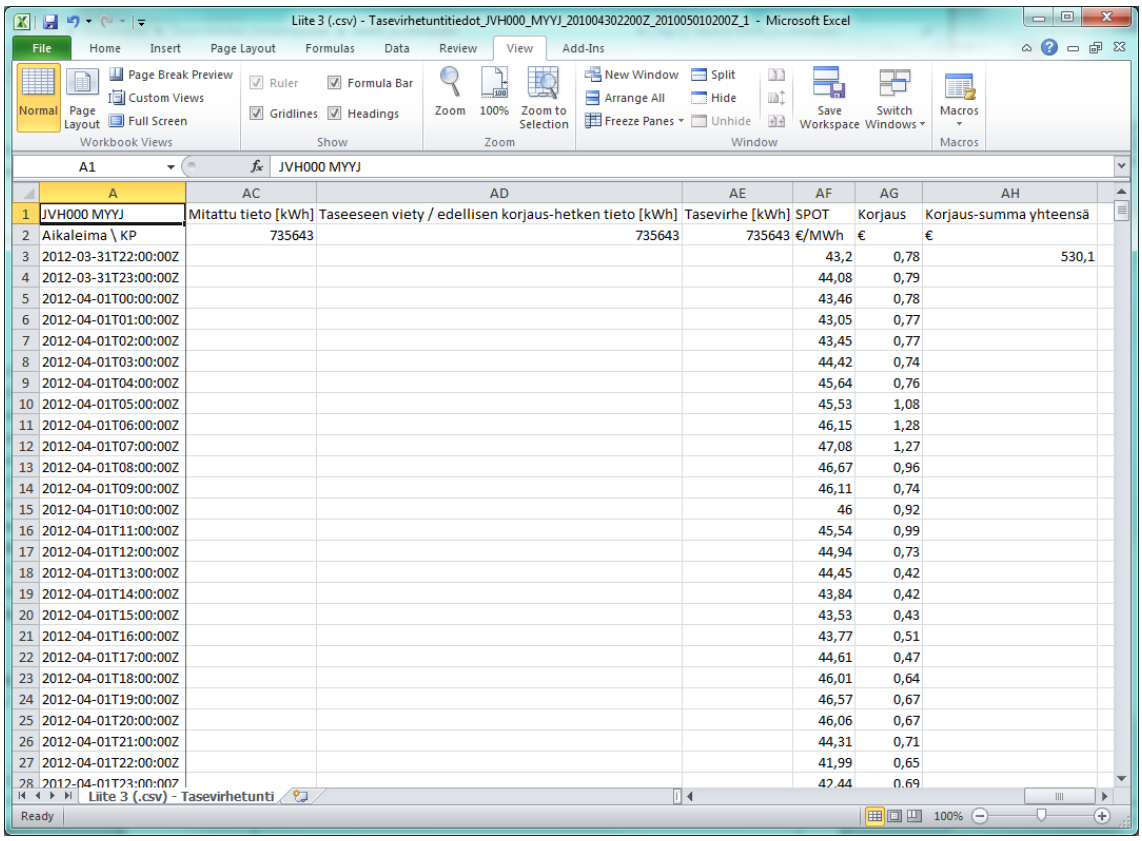
Taulukossa on keltaisella pohjalla merkitty tiedot, jotka täyttäjä hakee järjestelmistä vertailutaulukkoon. Valkoisella pohjalla on esitetty vertailun tulos.

Verkon järjestelmän tulee tuottaa ensimmäiselle välilehdelle sarakkeeseen A kaikki kyseisen myyjän hänen vastuualueellaan tuntimittauksen piirissä olevat käyttöpaikat tunnuksineen kasvavassa järjestyksessä. Sarakkeeseen B tulee ko. käyttöpaikan myyjätiedon voimaantulopäivämäärä ja sarakkeeseen C myyjätiedon päättymispäivämäärä. Voimassa olevilla sopimuksilla tämä kenttä jää tyhjäksi. Saman myyjän peräkkäiset sopimukset voidaan ketjuttaa samalle riville. Jos sopimuksien välissä on epäjatkuvuuskohta, jaetaan sopimukset eri riveille.

Sarakkeeseen D lasketaan vertailtavan ajanjakson, esimerkiksi vuosi tai tietty kuukausi (tarkistustonnan ollessa kyseessä), käyttöpaikkakohtainen ko. ajanjakson korjatusta tuntiaikasarjasta (ei välttämättä taseeseen viety) yhteenlaskettu sähköenergian käyttö.

Vastaavasti myyjän järjestelmä tuottaa oman näkemyksensä mukaisen vertailuaineiston välilehdelle "MYYJÄ". Tämä jälkeen näiden kahden välilehden toisiaan vastaavia soluja verrataan keskenään ja vertailun tuloksena saadaan järjestelmissä olevat eroavaisuudet.

Tasevirhetuntitiedot-raportin malliesimerkki



Liite 3 (.csv) - Tasevirhetuntitiedot_JVH000_MYYJ_201004302200Z_201005010200Z_1 - Microsoft Excel

	A	AC	AD	AE	AF	AG	AH
1	JVH000 MYYJ	Mitattu tieto [kWh]	Taseeseen viety / edellisen korjaus-hetken tieto [kWh]	Tasevirhe [kWh]	SPOT	Korjaus	Korjaus-summa yhteensä
2	Aikaleima \ KP	735643	735643	735643	€/MWh	€	€
3	2012-03-31T22:00:00Z				43,2	0,78	530,1
4	2012-03-31T23:00:00Z				44,08	0,79	
5	2012-04-01T00:00:00Z				43,46	0,78	
6	2012-04-01T01:00:00Z				43,05	0,77	
7	2012-04-01T02:00:00Z				43,45	0,77	
8	2012-04-01T03:00:00Z				44,42	0,74	
9	2012-04-01T04:00:00Z				45,64	0,76	
10	2012-04-01T05:00:00Z				45,53	1,08	
11	2012-04-01T06:00:00Z				46,15	1,28	
12	2012-04-01T07:00:00Z				47,08	1,27	
13	2012-04-01T08:00:00Z				46,67	0,96	
14	2012-04-01T09:00:00Z				46,11	0,74	
15	2012-04-01T10:00:00Z				46	0,92	
16	2012-04-01T11:00:00Z				45,54	0,99	
17	2012-04-01T12:00:00Z				44,94	0,73	
18	2012-04-01T13:00:00Z				44,45	0,42	
19	2012-04-01T14:00:00Z				43,84	0,42	
20	2012-04-01T15:00:00Z				43,53	0,43	
21	2012-04-01T16:00:00Z				43,77	0,51	
22	2012-04-01T17:00:00Z				44,61	0,47	
23	2012-04-01T18:00:00Z				46,01	0,64	
24	2012-04-01T19:00:00Z				46,57	0,67	
25	2012-04-01T20:00:00Z				46,06	0,67	
26	2012-04-01T21:00:00Z				44,31	0,71	
27	2012-04-01T22:00:00Z				41,99	0,65	
28	2012-04-01T23:00:00Z				42,44	0,69	